# PROGRAMACION CON EL ZX SPECTRUM Y ZX SPECTRUM+

W. Simister




EL ORDENADOR PERSONAL

ceac

# PROGRAMACION CON EL ZX SPECTRUM Y ZX SPECTRUM+






# PROGRAMACION CON EL ZX SPECTRUM Y ZX SPECTRUM+

**W. Simister**

ediciones ceac Perú, 164 - 08020 Barcelona - España

Traducción autorizada de la obra:
THE ART OF PROGRAMMING THE ZX SPECTRUM

Editado en lengua inglesa por
*Bernard Babani Ltd.*



ISBN 0-85934-0945


Perú, 164 - 08020 Barcelona (España)

Primera edición: Noviembre 1986

ISBN 84-329-7012-3

Depósito Legal: B-37254 - 1986

Impreso por
GERSA, Industria Gráfica
Tambor del Bruc, 6
08970 Sant Joan Despí (Barcelona)

*Printed in Spain*
*Impreso en España*

# INDICE

*Este libro está dedicado a mis padres,*
*los Sres. James,*
*a quienes debo el comienzo de mi programa.*

M. James

# PROLOGO

Cualquier programador, sea principiante o experto se hará probablemente muchas preguntas del tipo «¿cómo se programa el Spectrum para hacer tal o cual cosa...?» Este libro ha sido diseñado con el propósito de proporcionar algunas de las respuestas. Su objetivo es enseñar técnicas de programación que conviertan ese BASIC sencillo en programas realmente prácticos.

Este libro presenta todas las características del Spectrum necesarias para escribir programas de juegos divertidos y excitantes. El capítulo 2 presenta los gráficos de color de baja resolución. El capítulo 3 explica la forma de utilizar el generador de números aleatorios, para volver en el capítulo 4 al tema de los gráficos, esta vez de alta resolución. El capítulo 5 explora las capacidades sonoras del Spectrum. El tema del capítulo 6 son los gráficos dinámicos, mientras que el capítulo 7 analiza las potentes órdenes PEEK y POKE. El capítulo 8 está dedicado al mecanismo temporizador del Spectrum y en el capítulo 9 se analizan sus prestaciones para la manipulación de cadenas. Finalmente, en el capítulo 10 se analiza de nuevo el tema de los gráficos, esta vez a un nivel más avanzado.

Puede que alguien piense que una vez estudiado todo esto habrá agotado toda la información relativa al Spectrum. Pero nada más lejos de la realidad; quedan muchas cosas por aprender, mucho más de lo que cabría imaginar si se considera el tamaño compacto del Spectrum.

# 1. UN PRIMER CONTACTO CON EL SPECTRUM

El Sinclair Spectrum ofrece a sus usuarios una solución realmente maravillosa. Permite un uso versátil del color, ofrece gráficos de alta y baja resolución, y además añade sonido. El resultado es la posibilidad de escribir programas BASIC prácticos y excitantes, si se sabe cómo. El problema está en que para que el Spectrum haga cosas ingeniosas hay que poner en práctica algo más que lo que salta a la vista. Una cosa es haber aprendido a usar las órdenes del Spectrum, y otra muy diferente ser capaz de combinarlas en programas que hagan exactamente lo que se desea. Y ese es precisamente el objetivo de este libro: enseñar todos los secretos necesarios para hacer programas con el Spectrum.

## Una perspectiva excitante

En el curso de este libro se mostrarán detalladamente todas las características especiales del Spectrum, y, en tal sentido, en el Capítulo 2 se estudia el color del Spectrum y sus gráficos de baja resolución, en el Capítulo 4 se analizan sus

prestaciones relativas a los gráficos de alta resolución, y en el Capítulo 5 se define el sonido.

En la actualidad, muchos de los ordenadores domésticos tienen color y sonido. ¿Qué tiene entonces de especial el Spectrum? Bien, en primer lugar, el hecho de tener gráficos tanto de alta como de baja resolución es una gran ventaja. Generalmente los gráficos de baja resolución son más rápidos de utilizar y más prácticos, pero hay cosas, como dibujar formas y lineas de diseño delicado, que sólo se pueden hacer con gráficos de alta resolución; mediante la combinación de ambos tipos de gráficos en el mismo programa se puede crear una variedad de imágenes mucho mayor que utilizando sólo uno de los dos. En segundo lugar, el color del Spectrum es muy servicial. Hay algunas restricciones relacionadas con el uso del color en los gráficos de alta resolución, pero en los gráficos de baja resolución se pueden mezclar libremente los colores en la pantalla. En tercer lugar, el Spectrum utiliza una versión BASIC que posee algunas prestaciones muy útiles (que serán presentadas a lo largo de este libro) para escribir programas fácilmente al proporcionar todas las palabras reservadas. Incluso advierte muchos de los errores de programación en el momento de cometerlos; y esto es una gran ventaja.

## Programación ingeniosa

Este libro pretende ayudar a crear programas propios y precisamente por eso no constituye una introducción completa al BASIC o al Spectrum. Antes de seguir adelante, conviene estar acostumbrado a utilizar el teclado del Spectrum para escribir programas BASIC sencillos.

Se da por sentado que se ha profundizado en el manual que viene con el Spectrum y que se conoce el ordenador al nivel suficiente para no tener ningún problema de operación, y además, que se sabe lo suficiente para leer y escribir programas pequeños en BASIC. Pero puede ser que todavía no

se esté capacitado para trasladar las ideas propias a los programas BASIC reales. Esto es debido a que programar es algo más que conocer el BASIC, igual que hablar inglés es algo más que conocer unas cuantas palabras.

Aquí es donde encaja este libro. Intenta mostrar la forma de utilizar las técnicas de programación y las características del Spectrum para convertir las ideas en realidades. Dicho de otra forma, muestra la forma de construir programas en etapas muy fáciles, añadiendo características adicionales a medida que se aprende a utilizarlas. El libro contiene muchos retazos de programas pero también incluye más de veinte programas completos de juegos muy divertidos; después de todo, de vez en cuando habrá que tomarse un descanso en la programación, y está muy bien ser capaz de demostrar la inteligencia propia y la del Spectrum. Sin embargo, incluso los programas «completos» pueden ser perfeccionados; el programa perfecto no existe. Al jugar con los programas de juegos, conviene intentar ver nuevas formas de desarrollarlos más y poner en práctica las ideas propias. Es más fácil aprender añadiendo cosas a un programa existente que empezando desde cero.

## La diversión de aprender

La mayoría de los programas de este libro podrían ser descritos *groso modo* como programas de juegos. Una mirada rápida al índice confirmará la ausencia de programas «serios», lo cual puede resultar un poco sorprendente. Una razón para centrarse en los juegos radica en la posibilidad de diseñar programas de juegos que hagan entrar en escena todas las prestaciones del Spectrum. Otra razón es la posibilidad de convertir el proceso de aprendizaje en algo divertido, porque es más que probable que se aprenda más y mejor, si se está pasando un rato divertido. Es importante no ver los juegos como algo estúpido. La línea divisoria entre la diversión total y la búsqueda seria de conocimientos es muy borrosa en el campo de los ordenadores. Después de todo, los juegos espaciales de

hoy han sido desarrollados a partir de programas que originalmente sirvieron para poner en órbita hombres y naves espaciales y posarlos en la Luna.

## El sistema Spectrum

Este libro ha sido escrito utilizando un Spectrum de 16K, un aparato de televisión y un grabador de cinta casete, y las ilustraciones han sido hechas mediante una impresora ZX. Esto significa que los programas incluidos en este libro funcionarán en cualquier Spectrum, en uno de 16K o en el más poderoso de 48K. El Spectrum es un ordenador en color, pero incluso en el caso de utilizar una televisión en blanco y negro permite apreciar muchos de los efectos de las imágenes en color, que se verán en diferentes tonos de grises. No obstante, hay algunas combinaciones de colores, por ejemplo rojo y verde, que no se aprecian nada bien en blanco y negro; conviene por lo tanto ceñirse a los colores que contrastan bien. Una vez que se hayan introducido los programas de este libro o se hayan escrito programas propios, se pueden almacenar en cintas casete o, mediante un Microdrive Sinclair ZX, en micro-discos (floppy). Si se quieren probar los programas sin tener que escribirlos, es interesante saber que existe en el mercado una cinta casete de «Ramsoft» que incluye muchos de los programas completos de este libro; en la última página de esta obra se ofrecen más detalles.

# 2. GRAFICOS DE BAJA RESOLUCION

El Spectrum tiene una sola forma de presentar imágenes en la pantalla pero ofrece dos formas de alterar el contenido de la pantalla. Dicho de otra manera, el Spectrum utiliza el mismo método para presentar textos que para presentar gráficos. En términos generales, las dos formas de modificar la pantalla que ofrece al BASIC ZX dan lugar a dos formas diferentes de concebir los gráficos: alta resolución y baja resolución. Es importante entender que son simplemente formas diferentes de pensar; en realidad el Spectrum sólo puede generar imágenes en la pantalla de una forma.

## La pantalla del Spectrum

La pantalla del Spectrum esta constituida por una rejilla de puntos (256 horizontales por 176 verticales). Cualquiera de estos puntos puede estar «encendido» (on) o «apagado» (off). Mediante configuraciones de puntos encendidos o apagados se generan formas: cartas, dígitos, etc. Lo único que falta en esta descripción es el significado exacto de las palabras «encendido» y «apagado». Para simplificar las cosas, imaginemos un caso donde intervienen tan sólo dos colores, y para mayor simplicidad aún, aplicable al caso de no disponer de televisión

en color, imaginemos el blanco y negro. Si los puntos «encendidos» salen en la pantalla en negro y los puntos «apagados» salen en blanco, se puede ver que la descripción anterior tiene sentido. Si se forma la figura de una letra con puntos encendidos rodeados por puntos apagados, la letra será de color negro sobre fondo blanco. La similitud entre el empleo de puntos negros o encendidos para escribir sobre un fondo blanco y la escritura normal sugiere la utilización de los términos puntos de «tinta» (ink) y puntos de «papel» (paper) en vez de puntos encendidos y puntos apagados.

En resumen, la pantalla del Spectrum está constituida por dos tipos diferentes de puntos: puntos de tinta y puntos de papel. Generalmente, los puntos de tinta presentan un color y los puntos de papel otro. El Spectrum tiene dos órdenes que permiten seleccionar el color de los puntos de tinta y el de los puntos de papel. Las órdenes

```
INK 'color'
```

y

```
PAPER 'color'
```

asignan el color correspondiente al número encerrado entre comillas «color» a los puntos de tinta y de papel respectivamente. No es necesario recordar el número correspondiente a cada color porque los colores están escritos en la fila superior de las teclas numéricas del Spectrum. No obstante, son:

0 — Negro
1 — Azul
2 — Rojo
3 — Magenta
4 — Verde
5 — Cian
6 — Amarillo
7 — Blanco

Por lo tanto, la orden INK 2 hará que todos los puntos de tinta sean de color rojo y la orden PAPER 6 hará que todos los puntos de papel sean de color amarillo. La selección de los colores de la tinta y del papel es totalmente independiente. No hay nada que impida definir los mismos colores para el papel y para la tinta; por ejemplo INK 4 seguido por PAPER 4 definirán los puntos de tinta y de papel de color verde. Está claro que en este caso será imposible ver la diferencia entre los puntos de tinta y los del papel (los dos son del mismo color), pero es importante tener en cuenta que continúan siendo dos tipos de puntos diferentes.

El que haya seguido este análisis se preguntará cómo es posible que el Spectrum presente en pantalla más de dos colores al mismo tiempo. La respuesta radica en que la pantalla está dividida en numerosos cuadrados pequeños, llamados «posiciones de carácter», cada uno de sólo ocho puntos por ocho puntos. Es posible especificar independientemente el color de los puntos de tinta y de papel de cada posición de carácter. Esto puede parecer una idea un tanto extraña, pero de hecho facilita el manejo del Spectrum, porque el cuadrado de ocho por ocho es más que suficiente para presentar letras, dígitos y muchas de las formas pequeñas que se utilizan en los juegos. Con esta solución es muy cómodo trabajar con la pantalla en términos de posiciones de carácter, y ciertamente eso es exactamente lo que hace la instrucción PRINT. Cada vez que se utiliza una instrucción PRINT para imprimir un carácter en la pantalla, lo que se hace en realidad es definir qué puntos de un cuadrado de ocho por ocho serán puntos de papel y cuáles serán puntos de tinta. Se puede seleccionar el color de los puntos de papel y de tinta correspondiente a cada posición de carácter. La utilización de instrucciones PRINT para cambiar el contenido de las posiciones de carácter con objeto de hacer dibujos (quizás incluyendo texto) recibe el nombre de trazado de gráficos de baja resolución. El Spectrum tiene también un conjunto de órdenes que permiten convertir un solo punto de tinta a papel o viceversa. Estas órdenes nos llevan al campo de los gráficos de alta resolución. El único

problema radica en estar limitados todavía a un solo color de tinta y a un solo color de papel en cada posición de carácter, lo cual hace surgir algunos problemas interesantes a la hora de trabajar con gráficos de alta resolución «a todo color». Debido a estas dificultades y a otras, la mayoría de los programas del Spectrum consiguen sus efectos, en su mayor parte, mediante el empleo de gráficos de baja resolución. En el resto de este capítulo se analizan los detalles de los gráficos de baja resolución y en el capítulo 4 se estudia la otra cara de la moneda: los gráficos de alta resolución.

### *La orden PRINT — coma, punto y coma y apóstrofe*

La forma general de una instrucción PRINT es:

```
PRINT 'lista a imprimir'
```

donde «lista a imprimir» es un listado de conceptos a imprimir, separados por comas, por puntos y comas o por apóstrofes. Estos diversos separadores producen diferentes efectos en la colocación de los conceptos en la pantalla. Por ejemplo, el punto y coma da como resultado la ausencia de espacio en blanco entre los conceptos cuando son impresos en la pantalla. Es decir:

```
PRINT "a";"b"
```

imprimirá ab en la pantalla sin dejar ningún espacio entre las dos letras. Los efectos de estos tres separadores:

| *Separador* | | *efecto* |
|---|---|---|
| punto y coma | ; | ningún espacio entre conceptos impresos |
| coma | , | salto a la próxima zona de impresión antes de imprimir el siguiente concepto. |
| apóstrofe | ' | salto al principio de la próxima línea antes de imprimir el siguiente concepto. |

El único término de esta tabla que hay que explicar con mayor amplitud es «zona de impresión». La pantalla del Spectrum está dividida en dos zonas de impresión. La primera se extiende desde la columna 1 a la columna 16 y la segunda desde la 17 a la 32. Por lo tanto,

```
PRINT "a","b"
```

imprimirá a al principio de una linea y b en la columna 17.

Hay un punto adicional importante relativo a la utilización de los separadores. Si se finaliza una instrucción PRINT con *cualquier* separador, queda suprimido el comienzo usual de una línea nueva al final de la instrucción de impresión. Por ejemplo:

```
10 PRINT "a"
20 PRINT "b"
```

sitúa a y b al principio de dos líneas diferentes, pero

```
10 PRINT "a",
20 PRINT "b"
```

surte el mismo efecto que

```
10 PRINT "a","b"
```

La aplicación más importante de esta característica radica en la finalización de una instrucción PRINT con un punto y coma. Esto hace que la siguiente instrucción PRINT siga imprimiendo su primer concepto justo a continuación del último concepto de la primera instrucción PRINT. Por ejemplo:

```
10 PRINT "a";
20 GOTO 10
```

llenará la pantalla de letras A. Un efecto extraño de la norma

de que un separador situado al final de línea suprime el salto a una línea nueva se demuestra mediante

```
10 PRINT "a"'
```

La finalización de una instrucción PRINT con un apóstrofe no produce ningún cambio. La razón de ello hay que buscarla en el hecho de que la disposición de *cualquier* separador al final de la instrucción prohíbe el salto automático a una nueva línea. Por lo tanto, si se quiere permanecer en el espacio de una línea, utilizar PRITN o PRINT' y si se quiere saltar a una línea nueva utilizar PRINT''.

### *PRINT TAB y AT*

A pesar de que la mayoría de los problemas de impresión se pueden controlar mediante la utilización cuidadosa de la coma y del punto y coma, es difícil situar con precisión un concepto en un punto determinado de la pantalla utilizando simplemente esos separadores. Este problema se puede solucionar mediante la función TAB. Si se pone TAB(N) en una instrucción PRINT, el siguiente concepto a imprimir saldrá en la columna N de la línea en cuestión. En caso de que la instrucción PRINT haya traspasado la columna N, el primer concepto a imprimir saldrá en la columna N de la siguiente línea. Para comprobarlo, hacer la prueba con el programa siguiente:

```
10 PRINT TAB(25);"ab";AB(25);"ab"
```

Hay que tener en cuenta que, aunque es posible utilizar una coma después de la orden TAB, esta solución no resulta muy eficaz puesto que desplaza hacia adelante la posición de impresión desde donde la dejó la TAB. Si se utiliza un valor de N mayor que 32, el ordenador resta sucesivamente 32 hasta que llega a la gama correcta.

Todas las órdenes PRINT que se han utilizado hasta ahora

tienen una limitación en común: únicamente permiten colocar conceptos en la línea en cuestión. Pero hay una orden, PRINT AT, que permite imprimir cualquier concepto en cualquier sitio.

La orden AT es muy fácil de utilizar. La instrucción

```
PRINT AT Y,X;"PALABRA"
```

imprimirá PALABRA en la columan X de la línea Y. Si hay algo impreso con anterioridad en la columna X de la línea Y, no hay ningún problema, PALABRA lo sustituirá. La pantalla del Spectrum es de 32 caracteres de anchura por 22 líneas de altura. (La primera línea está en la parte superior de la pantalla y tiene asignado el número 0. La primera columna de caracteres está situada en el lado izquierdo de la pantalla y tiene asignado también el número 0). Esto significa que X debe estar comprendida entre 0 y 31 e Y entre 0 y 21. Si X o Y están fuera de esta gama, aparece en pantalla el mensaje de error «Entero fuera de gama» (Integer out of range). Un ejemplo sencillo de PRINT AT es:

```
10 PRINT AT 11,16;"*"
```

que imprimirá un asterisco en el centro de la pantalla. Cabe la posibilidad de utilizar PRINT AT para dibujar en la pantalla formas simples. Por ejemplo:

```
10 FOR i=0 TO 31
20 PRINT AT 10,i;"*"
30 NEXT i
40 FOR i=0 TO 21
50 PRINT AT i,15;"*"
60 NEXT i
```

imprimirá una línea horizontal y otra vertical de asteriscos. En una instrucción PRINT se pueden poner todas las AT que se quieran.

## Caracteres gráficos

Los ejemplos anteriores muestran la forma de utilizar la instrucción PRINT AT para dibujar formas en la pantalla, en vez de imprimir únicamente información en posiciones determinadas. No obstante, hacer formas con asteriscos y letras no es suficiente para crear buenos gráficos. Para solucionarlo, el Spectrum proporciona numerosos «caracteres gráficos» especiales. De ellos, veintiuno, que se estudiarán más adelante, pueden ser definidos por el usuario. Pero hay dieciséis caracteres gráficos pre-definidos que se pueden utilizar para hacer una gama limitada de formas sin que ello origine ningún problema adicional.

Los dieciséis caracteres gráficos están impresos en la fila superior del teclado. Concretamente, ocho caracteres están impresos en las teclas de la fila superior y los ocho restantes son simplemente sus «inversos»; las ocho formas negras se vuelven blancas y viceversa. Para obtener esos caracteres gráficos adicionales basta pasar al modo «gráficos» pulsando al mismo tiempo CAPS SHIFT y la tecla GRAPHICS (la tecla 9 de la hilera superior); la letra del cursor intermitente pasa a ser una «G» y si se pulsa luego cualquier tecla de la fila superior que tenga impresos caracteres gráficos sale en la pantalla ese carácter gráfico. Para imprimir los ocho caracteres restantes hay que pulsar también la tecla CAPS SHIFT mientras se está en el modo «gráfico». Es conveniente poner en práctica lo dicho para comprobar que se ha entendido la idea y que no hay problemas para imprimir los dieciséis caracteres gráficos. Es importante tener en cuenta que los ocho caracteres gráficos «inversos» son caracteres claramente diferentes de los ocho primeros. Esto puede parecer obvio, pero más adelante se verá que hay una forma de imprimir *cualquier* carácter como «inverso». Para salir del modo «gráfico» lo único que hay que hacer es pulsar la tecla GRAPHICS una vez más.

Debido a la dificultad de reproducir en papel caracteres gráficos (u otros) del Spectrum, se distinguirán los caracteres

gráficos colocando corchetes en torno a la palabra o dígito de la tecla que se debe pulsar para generar el carácter gráfico. Por ejemplo, [3] es el carácter gráfico que se obtiene pulsando la tecla «tres» mientras se está en el modo gráfico. Además se debería añadir una flecha superior «ˆ» para indicar «pulsar la tecla al mismo tiempo que se pulsa CAPS SHIFT». Por lo tanto, el carácter gráfico que se obtiene al pulsar la tecla «tres» y CAPS SHIFT estando en el modo «gráfico» es [ˆ3].

Debido al número limitado de las formas correspondientes a los dieciséis caracteres gráficos, no resultan tan prácticos como cabría esperar. Se pueden utilizar para dibujar formas como cuadrados etc. imprimiéndolos (PRINT) en la combinación adecuada, pero su principal aplicación radica en la creación de formas sólidas y pequeñas para juegos, etc. Por ejemplo, si se quiere imprimir la forma de un «perro» en la pantalla, probar con:

```
10 PRINT "[1][ˆ1][ˆ3]"/"[8][ˆ5][5]"
```

En el capítulo 6 se describe el procedimiento para mover el perro por la pantalla.

El problema de los dieciséis caracteres gráficos pre-definidos es que si se quiere dibujar una forma, por ejemplo un cuadrado o un círculo, normalmente se puede conseguir más fácilmente mediante los gráficos de alta resolución, y si se quiere dibujar una forma sólida como la del «perro», es mejor hacerlo mediante los caracteres gráficos «definidos por el usuario» (que se describen más adelante). A pesar de lo dicho anteriormente, merece la pena mirar los caracteres gráficos pre-definidos para ver si alguno de ellos puede realizar el trabajo. Por ejemplo, son particularmente prácticos para dibujar lineas anchas horizontales o verticales. Probar con:

```
10 FOR i=1 TO 32
20 PRINT "[3]";
30 NEXT i
```

```
40 FOR i=1 TO 31
50 PRINT TAB (16);"[5]"
60 NEXT i
```

Obsérvese la utilización de la TAB y del punto y coma para situar los caracteres gráficos en la pantalla.

### *CHR$*

Hay otra forma de generar caracteres gráficos o cualquier otro tipo de caracteres: la función CHR$. Si todos los caracteres del Spectrum estuvieran escritos en orden, se podría escoger un carácter diciendo algo así como «el carácter 36º». Esto es exactamente lo que hace la función CHR$. CHR$(36) es el carácter 36º en el conjunto de caracteres del Spectrum. Si se quiere ver todo el conjunto de caracteres del Spectrum, probar con:

```
10 FOR i=32 TO 255
20 PRINT CHR$(i);
30 FOR j=1 TO 100
40 NEXT j
50 NEXT i
```

Se observará que algunas veces sale impresa únicamente una letra y otras veces una palabra completa, por ejemplo COPY o PRINT. El Spectrum trata a todas las palabras BASIC que se escriben con una sola pulsación como un solo símbolo.

La característica más importante de la función CHR$ radica en que proporciona una conexión entre números y caracteres. Como un ejemplo de la forma de empleo de esta función, ejecutar el programa siguiente:

```
10 PRINT CHR$(INT(RND*16+128));
20 GOTO 10
```

La pantalla se llenará de caracteres gráficos aleatorios. El funcionamiento correcto de este programa quedará claro después del próximo capítulo, donde se explica la función RND$. No obstante, el papel de la función CHR$ de convertir números a caracteres tiene que estar claro si se intenta ejecutar ese mismo programa pero suprimiendo la función CHR$.

## Caracteres gráficos definidos por el usuario

No hay ninguna duda de que la característica que hace prácticos los gráficos de baja resolución del Spectrum es la posibilidad de definir nuevos caracteres. Esto es relativamente fácil de hacer y aumenta en gran medida el nivel de detalle de los gráficos de baja resolución. No sólo es fácil definir nuevos caracteres sino que, además, una vez que están definidos se pueden utilizar como cualquier otro carácter existente.

La siguiente pregunta es: ¿Dónde se encuentran estos caracteres adicionales? Si se entra en el modo «gráfico» y se pulsa cualquiera de las teclas de la A a la U del alfabeto causará sorpresa ver impresas en la pantalla las letras correctas de la A a la U. Es como si hubiera dos formas de imprimir estas letras: de la forma normal y en el modo gráfico. Esto es debido a que este conjunto de teclas tiene asignadas inicialmente las formas de las letras del alfabeto. El tema de este apartado es la forma de cambiar su definición por algo más útil.

El primer problema que se debe solucionar es cómo especificar la forma correspondiente al carácter en términos de puntos de tinta y puntos de papel. Una manera de hacerlo es representar los puntos de papel mediante ceros («0») Y los puntos de tinta mediante unos («1»). Por lo tanto, si se quiere definir un «perro» se podría utilizar la siguiente configuración de ceros y unos.

```
1 1 0 0 0 0 0 1
1 1 0 0 0 0 1 0
0 0 1 1 1 1 1 0
0 0 1 1 1 1 1 0
0 0 1 1 0 1 1 0
0 1 1 0 0 0 1 1
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

Hay que tener en cuenta que un carácter utiliza un recuadro de ocho por ocho puntos. Si resulta difícil ver la forma de un perro en el cuadro de ceros y unos anterior, copiarlo y colorear en negro los unos. Una vez que se ha entendido el método de componer una configuración de ceros y unos, se está en disposición de definir la forma que se quiera.

El siguiente problema es transferir esta definición al carácter definido por el usuario. Tal como se ha mencionado anteriormente, los caracteres definidos por el usuario están relacionados con las teclas «a» a la «u». La definición de los caracteres que estas teclas generarán cuando se pulsen en el modo gráfico, se almacena en la memoria del Spectrum, una fila cada vez. Para cambiar la definición de una fila de un carácter concreto hay que utilizar una instrucción muy complicada. Por ejemplo, si se quiere cambiar la definición de la tercera fila de la letra «B» hay que utilizar:

```
POKE USR "b"+2,BIN xxxxxxxx
```

donde las xxxxxxxx son sustituidas por la fila de ceros y unos que se quiera. No es necesario entender el funcionamiento de esta línea BASIC para definir caracteres nuevos. La única cosa que debe tenerse presente es que las filas están numeradas de 0 a 7, por lo que en el ejemplo anterior la tercera fila se modifica poniendo «b+2» después de USR. En términos generales, para cambiar la enésima fila del carácter C por XXXXXXXX, utilizar:

```
POKE USR"C"+n-1,BIN xxxxxxxx
```

Es evidente que para definir un carácter completo habrá que definir las ocho filas nuevas.

Aunque no es necesario, es interesante comprender qué hace la instrucción anterior para definir caracteres nuevos. (En una primera lectura se puede omitir este párrafo). La función USR «letra» da la dirección de la posición de memoria que se utiliza para almacenar la primera fila del carácter gráfico correspondiente a «letra». Por lo tanto,

```
USR "letra"+1
```

da la dirección de la segunda fila, y así sucesivamente. La función BIN convierte la fila de ceros y unos en un número. Para verlo, probar con:

```
10 PRINT BIN 011
```

y diferentes filas de ceros y unos. La conversión se hace considerando a la fila como un número binario; de ahí el nombre de la función. Después, mediante la orden POKE se almacena el número resultante en la posición de memoria correcta. En el capítulo 7 se estudia este tema. El número que se almacena en la posición de memoria correcta genera la configuración de puntos de tinta y de papel correspondiente cuando el Spectrum imprime el carácter gráfico.

Como ejemplo de carácter gráfico definido por el usuario, volvamos a nuestro perro:

```
10 POKE USR "a"+0,BIN 11000001
20 POKE USR "a"+1,BIN 11000010
30 POKE USR "a"+2,BIN 00111110
40 POKE USR "a"+3,BIN 00111110
50 POKE USR "a"+4,BIN 00110110
60 POKE USR "a"+5,BIN 01100011
70 POKE USR "a"+6,BIN 00000000
80 POKE USR "a"+7,BIN 00000000
90 PRINT"[A]"
```

Este programa define en primer lugar el nuevo carácter, en las líneas 10 a 80, y luego lo imprime, en la línea 90. Una vez ha sido ejecutado el programa, la nueva definición de [A] permanece hasta que se desconecta el Spectrum o hasta que se la vuelva a definir. Para comprobarlo basta pulsar la tecla «A» mientras se está en el modo gráfico.

No hay ningún problema para imprimir naves espaciales, cohetes, barcos, tanques, etc. En el resto de este libro se volverán a tratar, una y otra vez, los gráficos definidos por el usuario.

## La utilización del color

El método básico de control del color ya ha sido presentado: las órdenes INK y PAPER. No obstante, hay algunas características sobre su funcionamiento exacto que no han sido tratadas. La primera es que una orden INK o PAPER selecciona el color de todos los puntos de tinta y de papel impresos *después* de la orden. Es decir, INK y PAPER no afectan al color de los puntos que se encuentran ya en la pantalla. Por ejemplo, hacer la prueba con:

```
10 INPUT "¿color de papel?";p
20 INPUT "¿color de tinta?";i
30 PAPER p
40 INK i
50 FOR j=1 TO 32
60 PRINT "X";
70 NEXT j
80 GOTO 10
```

Después de introducir el color de papel y de tinta saldrá una línea completa de letras X de los colores especificados. Se debe tener en cuenta que aunque cada línea nueva de la pantalla sale con los colores nuevos, las líneas anteriores no sufren ninguna alteración. (Si se trabaja con una pantalla en blanco y negro, los colores saldrán en matices grises). Mientras se

ejecuta este programa, merece la pena probar los efectos de especificar el mismo color de tinta y de papel. No hay que olvidar que, aunque no se vean, las X siguen estando ahí.

Este método de controlar el color es muy fácil de usar, pero es un tanto embarazoso si se quieren imprimir en la misma línea muchas cosas de colores diferentes. Esto se puede hacer poniendo órdenes INK y PAPER en la instrucción PRINT. Por ejemplo, la línea

```
10 PRINT INK 0;PAPER 7;"X";INK 7;PAPER 0;"X"
```

imprime una X negra sobre fondo blanco y luego una X blanca sobre fondo negro. Lo más importante que hay que comprender con respecto a la utilización de las órdenes INK y PAPER en las instrucciones PRINT es que sus efectos son temporales; duran únicamente hasta el final de la instrucción PRINT. Al final de la orden PRINT los colores de papel y de tinta vuelven a ser los que eran antes de su ejecución.

Hay otras dos órdenes esenciales para la utilización del color: CLS y BORDER. La orden CLS simplemente borra la pantalla, pero merece la pena concretar con exactitud qué es lo que hace: convierte todos los puntos de la pantalla en puntos de papel, lo que significa que toda la pantalla adquiere el color seleccionado por PAPER. La orden BORDER simplemente selecciona el color del margen que rodea a la zona de la pantalla en la que el Spectrum puede imprimir. La única característica a resaltar con respecto a la orden BORDER es que además controla el color de la zona de la pantalla donde va a parar la entrada. A título de ejemplo de la forma de empleo de las órdenes CLS y BORDER, probar con:

```
10 FOR i=0 TO 7
20 PAPER i
30 CLS
40 BORDER 7-i
50 PAUSE 5
60 NEXT i
```

Si se quiere ver cambiar los colores con más lentitud, aumentar el número que sigue a la orden PAUSE.

Como ejemplo final de la utilización de los colores, el programa siguiente imprime una serie de franjas de colores.

```
10 FOR c=0 TO 7
20 PAPER c
30 PRINT "    ";:REM 4 espacios
40 NEXT c
50 GOTO 10
```

### *INVERSE Y OVER*

La característica especial de las órdenes INVERSE y OVER es que ambas afectan a la forma de situar los caracteres en la pantalla. La orden INVERSE 1 hace que todos los caracteres siguientes salgan en la pantalla con puntos de tinta sustituyendo a los puntos de papel y viceversa. Para volver al modo normal de operación es necesario usar la orden INVERSE 0. Hay que tener en cuenta que la orden INVERSE no genera ningún carácter nuevo ni cambia ninguna definición existente; simplemente intercambia los puntos de papel y de tinta cuando se imprime el carácter en la pantalla. Para comprobar los efectos de la orden INVERSE, ejecutar:

```
10 INVERSE 1
20 IMPUT a$
30 PRINT a$
40 GOTO 20
```

Después de ejecutar este programa introducir INVERSE 0 para volver a la normalidad.

La orden OVER 1 también cambia la forma de presentación de los puntos de papel y tinta en la pantalla, aunque

de una manera más difícil de describir que en el caso de INVERSE. Después de OVER 1, lo que aparece en la pantalla cuando se imprime un carácter depende de lo que ya había en la pantalla. La regla es que si el punto anterior y el punto nuevo son idénticos, el resultado es un punto de papel, pero si los dos puntos son diferentes el resultado es un punto de tinta. Esto puede parecer complicado, pero básicamente significa que si se imprime un carácter nuevo en el mismo lugar que un carácter ya existente, ambos aparecen en la pantalla, pero donde se cruzan se «anulan». Por ejemplo:

```
10 CLS
20 OVER 1
30 PRINT AT 0,0;"A"
40 PRINT AT 0,0;"—"
50 PRINT AT 1,0;"A"
60 PRINT AT 1,0;"\"
70 OVER 0
```

La primera instrucción sitúa la letra A en la esquina superior izquierda. La segunda instrucción PRINT imprime en el mismo lugar un carácter subrayado. Debido a la orden OVER 1 de la línea 20, ambos caracteres aparecen juntos. No obstante, las líneas 50 y 60 imprimirán la letra A y \ en el mismo lugar, y a pesar de que ambas aparecen en la pantalla al mismo tiempo, si se mira cuidadosamente se comprobará que en el lugar donde se cruzan no hay puntos de papel.

Una aplicación muy especial de la orden OVER es la impresión y borrado de un carácter aislado. Por ejemplo, si se imprime un mensaje en la pantalla, se le puede hacer desaparecer imprimiéndolo por segunda vez después de la orden OVER 1.

```
10 CLS
20 PRINT AT 0,0;"un mensaje"
30 OVER 1
40 GOTO 20
```

Si se ejecuta este programa se observará que el mensaje aparece y desaparece de la pantalla intermitentemente. Los efectos de OVER 1 e INVERSE 1 permanecen hasta que son contrarrestados por OVER 0 o INVERSE 0 o hasta que se desconecta el ordenador. Por lo tanto ¡cuidado! Al igual que INK y PAPER, se pueden usar también INVERSE y OVER en las instrucciones PRINT; sus efectos serán entonces temporales. Las órdenes INVERSE y OVER aparecerán de nuevo en el capítulo dedicado a los gráficos de alta resolución.

### *BRIGHT y FLASH: atributos*

Las dos últimas órdenes de generación de caracteres en la pantalla tienen en común con INK y PAPER el no afectar en forma alguna a la configuración de puntos de tinta y papel que sale en la pantalla. En cambio, sí alteran la forma de presentación de los puntos de tinta y papel en la pantalla. Después de BRIGHT 1, todos los puntos de papel y tinta tienen un nivel mayor de brillo. Después de FLASH 1, todos los puntos de papel y tinta son intermitentes (alternan entre el color de tinta y el de papel). Una vez más, para anular dichos efectos es necesario usar BRIGHT 0 y FLASH 0. Para ver los efectos de estas dos órdenes, probar con:

```
10 CLS
20 PAPER 1
30 INK 7
40 PRINT “Esto es normal”
50 BRIGHT 1
60 PRINT ”Esto es brillante”
70 BRIGHT 0
80 FLASH 1
90 PRINT “Esto es intermitente”
100 BRIGHT 1
110 PRINT “Esto es brillante e intermitente”
120 BRIGHT 0
130 FLASH 0
```

Obsérvese que para imprimir caracteres intermitentes y brillantes se pueden usar conjuntamente las órdenes BRIGHT y FLASH.

Cuando se imprime un carácter en la pantalla, lo único que determina su aspecto es la configuración de puntos de tinta y papel y la forma en que los puntos individuales de papel y tinta salen en la pantalla. Las dos únicas formas de influir en la configuración de puntos que sale en la pantalla son: la definición de nuevos caracteres gráficos y el empleo de las órdenes INVERSE y OVER. Las órdenes INK, PAPER, FLASH y BRIGHT afectan, todas ellas, a la forma en que salen en la pantalla los puntos individuales. De hecho, el Spectrum manipula la imagen de la pantalla de una forma escalonada. Una zona de memoria se usa para almacenar la configuración de puntos y otra zona de memoria contiene la forma en que se debe presentar cada punto. La forma de presentación de cada punto en la pantalla recibe el nombre de sus «atributos»; la segunda zona de memoria es por lo tanto la «zona de atributos». En el capítulo 10 se volverá a tratar el tema de los atributos.

**El atributo 8 y el color 9**

Se da frecuentemente el caso de querer imprimir algo en la pantalla sin alterar los atributos de la posición de carácter. Si se sabe cuáles son, no hay ningún problema; lo único que hay que hacer es usar la lista correcta de órdenes de atributos; por ejemplo, si se sabe que una posición es intermitente, poner FLAH 1 antes de imprimirla. Pero si no se sabe cuáles son los atributos habrá problemas, a menos que se conozca el atributo 8. Poniendo un 8 en cualquiera de las órdenes de atributos (BRIGHT, FLASH, INK, PAPER), se consigue que cualquier carácter que se añada a la pantalla retenga los atributos del carácter al que sustituye. Por ejemplo, después de FLASH 8 cualquier carácter nuevo que se imprima brillará intermitentemente si ocupa una posición que era anterior-

mente intermitente, y sino estará estacionario. Después de PAPER 8 cualquier carácter nuevo tendrá el color de papel del carácter que sustituye. Este tipo de comportamiento hace que el atributo 8 reciba el nombre de «transparente» porque deja ver el atributo original.

Un problema similar surge a veces con los colores de tinta y papel. Si se quiere imprimir algo en la pantalla con el mismo color de papel, ¿qué color de tinta se debe usar para tener la seguridad de que resaltará? Por ejemplo, si el color de papel actual es el negro, está claro que no se va a imprimir en tinta negra. Si se conoce el color de papel actual, no hay ningún problema; basta elegir un color de tinta que contraste. Pero si no se conoce o si cambia, sí hay problemas. Este problema se soluciona utilizando el código de color «falso» 9, que simplemente selecciona un color que contraste con el color en vigor. Por ejemplo, INK 9 seleccionará el color negro si el color de papel actual es claro (4 a 7), y el blanco si el color de papel actual es oscuro (0 a 3). Lo mismo sucede con PAPER 9, pero en este caso el color es seleccionado por contraste en el color de tinta actual. Con el código de «color» 9 se puede tener siempre la seguridad de que las cosas resaltarán en la pantalla, pero se debe tener presente que en la práctica el código 9 es o blanco o negro, lo cual puede producir imágenes muy grises.

## Resumen

En este capítulo se han presentado las formas básicas de manipular la pantalla del Spectrum. Aunque los programas presentados no son excesivamente impresionantes, es muy difícil avanzar sin conocer este punto; además, las técnicas aplicables a los gráficos de baja resolución volverán a aparecer y serán desarrolladas en el resto de este libro.

# 3. LA DIVERSION DEL AZAR

En el sentido abstracto, el azar es un tema bastante esotérico, por lo cual puede resultar sorprendente verlo incluido en un libro de juegos de programación. Sin embargo, si se piensa detenidamente, el azar o suerte es un componente fundamental en todo tipo de juegos. En primer lugar, hay juegos de azar (juegos de cartas y de dados); también hay juegos en los que cuenta la rapidez con que se reacciona ante una situación fortuita; y por último, hay juegos en los que se pone a prueba la habilidad personal para derrotar a un oponente cuyas decisiones no se pueden predecir por adelantado. Los números aleatorios son la base de todos estos juegos.

## Seudo-aleatoriedad

¿Qué es un número aleatorio? La respuesta ya ha sido sugerida anteriormente; es un número imposible de predecir por adelantado. El resultado del lanzamiento de una moneda al aire está sometido al azar, al igual que el lanzamiento de un dado; No se puede saber con anticipación si la moneda saldrá «cara» o «cruz» y tampoco se puede saber qué cara del dado saldrá ni hay forma alguna de controlar el resultado. El hecho

de que los jugadores no puedan influir en el resultado es el aspecto importante de una situación aleatoria en lo que se refiere a los juegos.

¿Puede un ordenador generar un número aleatorio? Después de todo, un ordenador únicamente puede dar salida a una función de lo que se le ha introducido anteriormente. En este caso, los escépticos tienen razón; un ordenador no puede generar un número verdadero aleatorio. Un ordenador únicamente puede generar los números que calcula. La característica más importante que se utiliza en el azar es la de no poder predecir el siguiente resultado o número. Se puede hacer que el ordenador calcule un conjunto de números de tal forma que sea muy difícil predecir qué número será el siguiente. Los números de estas características se llaman «seudo-aleatorios».

Para clarificar el asunto, hay que decir que un número seudo-aleatorio no es consecuencia de una situación de azar (como el lanzamiento de un dado). Por lo tanto, es en teoría predecible, pero se puede generar, a efectos prácticos, de tal forma que ningún observador pueda descubrir cual será. En este sentido, se podría decir que el ordenador genera números impredecibles, no números aleatorios.

Un ordenador genera sus números aleatorios mediante una formula, por lo que cualquier persona que tenga una copia de la fórmula puede predecir el siguiente número de la serie. Pero en la práctica, la fórmula es suficientemente complicada que, para los juegos, donde todo sucede con rapidez, habría que ser un genio de las matemáticas para aplicar la fórmula a tiempo.

### *RND y RAND*

El Spectrum se sirve de la función RND para generar números seudo-aleatorios en la gama del 0 al 1. Cada vez que se incluye la palabra RND, el Spectrum calcula el siguiente

número de la secuencia. Ahora ya no es tan preocupante la idea de que el siguiente número de la secuencia pudiera ser calculado. Los números que calcula RND tienen otra característica muy importante: todos los números comprendidos entre 0 y 1 tienen aproximadamente las mismas posibilidades de ser generados. Otra forma más técnica de decirlo es: la orden RND genera números seudo-aleatorios «distribuídos uniformemente» entre 0 y 1.

Ahora que sabemos lo que pasa cuando se usa la función RND, veamos la forma de utilizarla. Para mostrar el tipo de salida que se obtiene cuando se solicita un número aleatorio, introducir.

```
10 PRINT RND
20 GOTO 10
RUN
```

Se obtendrá una pantalla de números, todos comprendidos entre 0 y 1. Por ejemplo:

```
·0011291504
·08581543
```

¿Hay alguna coincidencia sorprendente? ¿Se pueden predecir los dos números de la parte superior de la pantalla? desconectar el ordenador, volverlo a conectar e intentarlo de nuevo. Ahora la pantalla presentará un listado que comienza con esos mismos números. La razón de que esto suceda así es que la fórmula que rige el orden de los números aleatorios es la misma en todos los ordenadores Spectrum; en el manual se incluye la formula, de modo que no es ningún secreto. Si se ejecuta el programa (RUN) más de una vez sin desconectar el ordenador entre intervalos, la secuencia continúa desde el lugar en el que estaba antes.

El hecho que el orden aleatorio sea totalmente repetible es en realidad muy eficaz para algunas aplicaciones (por ejem-

plo, para probar simulaciones alternativas, donde se quiere repetir el mismo modelo de situaciones aleatorias), pero para otros propósitos es completamente ineficaz. Ejecutar juegos de azar con el Spectrum perdería pronto su atractivo si no fuera por la función RAND. (En realidad, cuando se pulsa la tecla RAND el Spectrum presenta en la pantalla la palabra RANDOMIZE, pero para facilitar la escritura de los programas utilizaremos RAND en lugar de RANDOMIZE). La función RAND da una instrucción al ordenador: El punto de la secuencia donde hay que comenzar; puede ser un punto seleccionado o un punto al azar, como si se clavara un alfiler en una lista. Para predeterminar el punto de partida, escribir.

```
RAND cualquier número
```

Por ejemplo, ejecutar:

```
10 RAND 35
20 PRINT RND
30 GOTO 20
```

Ejecutarlo varias veces. La secuencia es siempre la misma. Si se prueba con:

```
10 RAND 35
20 PRINT RND
30 GOTO 10
```

se verá que sigue imprimiendo el mismo número: el punto de partida de la secuencia proporcionado por RAND 35.

Para obtener una secuencia diferente cada vez, incluir

```
10 RAND 0
```

Cuando se usa RAND 0, lo que en realidad sucede es que el ordenador usa el valor de un contador interno que cuenta el número de imágenes televisivas presentadas en la pantalla,

desde que se conectó el ordenador (ver el capítulo 8). Aunque el lugar donde comienza la secuencia en realidad corresponde al tiempo que el ordenador lleva conectado, el hecho de que el contador opere a una velocidad de 50 pasos por segundo significa que es prácticamente imposible predecir su posición cuando se escribe RAND 0. En resumen: la función RND calcula el siguiente número aleatorio de la secuencia; se puede utilizar RAND para seleccionar el punto de partida de la secuencia, de una forma lo más aproximada al azar que podemos conseguir. Probarlo ejecutando:

```
10 RAND 0
20 PRINT RND
30 GOTO 10
```

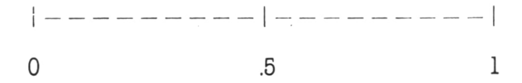
Imprimirá un conjunto de puntos de partida seleccionado por RAND 0. Se debe ver que a pesar de que el valor de partida aumenta con el tiempo, es casi imposible predecirlo. Así, la máxima aleatoriedad, es decir, series impredecibles de números, que el Spectrum puede producir, se obtiene mediante:

```
10 RAND 0
20 PRINT RND
30 GOTO 20
```

**Para hacerlo realidad**

A primera vista, la cadena de números que sale en la pantalla al solicitar un número aleatorio no parece demasiado útil. Pongamos una aplicación sencilla y veamos la forma de obtener el tipo de resultados que buscamos. Imaginemos el lanzamiento de una moneda al aire. Existen dos posibilidades, «cara» o «cruz». ¿Cómo simplificar la salida en bruto de la función RND para obtener una de las dos respuestas, y que sea imparcial, es decir con las mismas probabilidades de que la moneda salga «cara» o salga «cruz»? La solución es dividir

la línea de respuestas en dos mitades exactamente iguales. Como la línea va de 0 a 1, es fácil. El punto central será 0,5:




Como los números generados por RND tienen las mismas probabilidades de caer en cualquier punto de la línea, la mitad caerá debajo de 0,5, y la otra mitad caerá por encima. Si a los números que caen por debajo de 0,5 se les llama «cara» y a los que caen por encima «cruz», lógicamente saldrán tantas caras como cruces. Al traducirlo en un programa se obtiene:

```
10 RAND 0
20 LET r=RND
30 IF r<.5 THEN PRINT "cara"
40 IF r>=.5 THEN PRINT "cruz"
50 GOTO 20
```

La línea 10 elige al azar el punto de comienzo de los números aleatorios. La línea 20 pone un número aleatorio en r y las líneas 30 y 40 deciden en qué mitad de la gama cae el número. Si es inferior a 0,5 imprimimos «cara» y si es superior o igual a 0,5 imprimimos «cruz». Así de fácil. En realidad, este programa de lanzamiento de monedas al aire no es excesivamente brillante, así que más adelante trataremos este problema.

¿Y si se tira una moneda defectuosa, una que de cada cuatro veces, tres sale cara? No resulta difícil ver la forma de alterar el programa para obtener los mismos resultados que con esa moneda: basta transformar la división de la línea en dos partes desiguales. En general, si la probabilidad de que salga cara es P:

```
10 RAND 0
20 INPUT p
30 LET r=RND
```

```
40 IF r<p THEN PRINT "cara"
50 IF r>=p THEN PRINT "cruz"
60 GOTO 30
```

Para tirar una moneda lo único que hay que hacer es permitir dos posibilidades, cada una de las cuales sucede con la misma probabilidad, a menos que deliberadamente se alteren las probabilidades como en el ejemplo de la moneda trucada. Hay otras situaciones aleatorias en las que existe un número mayor de posibilidades y las probabilidades de diferentes resultados no son idénticas.

Para el observador medio, el tiempo meteorológico en Inglaterra parece ser con frecuencia una cuestión de suerte, o más bien de mala suerte. Vamos a escribir un programa para ver si los pronósticos del hombre del tiempo, basados en principios científicos, son más certeros que los hechos en base a una «adivinanza razonable». La adivinanza combina un elemento aleatorio con el conocimiento de las pautas meteorológicas propias de la estación climatológica. El siguiente programa corresponde a la primavera. En los cien días de primavera (el período que va desde la mitad de febrero a la mitad de mayo) son de esperar unos 40 días de sol, 30 días de nubes, 20 días de lluvia y 10 días de nieve. Veamos la forma de encajar todo esto. Imaginemos una línea con el 0 en un extremo y el 1 en el otro y marquemos secciones de una longitud proporcional a la probabilidad de cada tipo de tiempo meteorológico.

Por ejemplo:

```
          Sol             Nubes        Lluvia   Nieve

          .4               .3            .2       .1

|- - - - - - - - - -|- - - - - - - -|- - - - -|- - -|

0                   .4              .7        .9    1
```

Si se generan números aleatorios equitativamente entre 0 y 1, la probabilidad de que un número caiga en una sección determinada es proporcional a la longitud de la sección. Esta es la clave para seleccionar el tiempo meteorológico con la probabilidad correcta. La condición meteorológica correspondiente a la sección en la que cae el número aleatorio es la que se pronostica. Así, al número 0,6712348117 le corresponde «Nubes».

Pero hay un problema con este sistema. ¿Qué sucede si el número aleatorio es precisamente uno de los puntos de separación de las diferentes secciones, por ejemplo el 0,4; será un día de sol o de nubes? No importa que salga una cosa u otra, lo importante es que salga algo. En realidad, lo correcto es asignar el punto 0 a la primera sección y seguir asignando los puntos-frontera a la sección de la derecha. Pero ¿qué sucede con el punto correspondiente al 1? Bien, si se analiza cuidadosamente la definición de RND se verá que da números a partir del 0 *sin incluir* el 1. Por lo tanto, no nos debe preocupar el punto correspondiente al 1, ya que no sale nunca. El programa del pronóstico meteorológico debería ya resultar obvio:

```
10 RAND 0
20 LET r=RND
30 PRINT "Predicción del tiempo"
40 IF r<.4 THEN PRINT "sol"
50 IF r>=.4 AND r<.7 THEN PRINT "nubes"
60 IF r>=.7 AND r<.9 THEN PRINT "lluvia"
70 IF r>=.9 THEN PRINT "nieve"
```

La única parte complicada de este programa consiste en determinar la sección a la que pertenece el número aleatorio, aunque no debería plantear problemas si se consulta el diagrama anterior. Hay muchas formas ingeniosas de hacerlo; algunas ahorran memoria y otras son más rápidas, pero la utilizada anteriormente es la más fácil de entender y funciona en cualquier ordenador.

### Enteros aleatorios

Dividir el intervalo comprendido entre 0 y 1 es una forma de seleccionar cuál es la «acción» que va a ocurrir, pero esa no es la única forma. En un caso donde ocurre con igual probabilidad un número determinado de acciones, hay una alternativa: multiplicar el número generado por el ordenador por el número de posibilidades, redondearlo a un número entero y sumar uno a la respuesta. En realidad, es más fácil de lo que parece. Veamos el ejemplo práctico del lanzamiento de un dado. Hay que elegir una de 6 posibilidades. Se podría aplicar el método de dividir la línea en 6 partes iguales, pero probemos el nuevo método. Si se multiplica RND por 6 se obtendrá un número que está comprendido entre 0 y menos de 6. Si se usa la función INT para convertir el número en un número entero se obtiene un número comprendido entre el 0 y el 5, y sumando 1 se obtiene un número comprendido entre el 1 y el 6. Para verlo en acción, ejecutar el siguiente programa

```
10 RAND 0
20 LET r=RND
30 PRINT r
40 LET r=r*6
50 PRINT r
60 LET r=INT(r)
70 PRINT r
80 LET r=r+1
90 PRINT r
```

Si se ejecuta este programa varias veces se estará en disposición de ver lo que está sucediendo. Por supuesto, en la práctica se realizaría todo el procedimiento con una sola instrucción:

```
10 RAND 0
20 LET r=INT(RND*6)+1
30 PRINT r
40 GOTO 20
```

En términos generales, cuando se quieran generar números aleatorios comprendidos entre n y m, hacerlo con:

```
10 LET r=INT(RND*(m-n+1))+n
```

Normalmente n es 1, luego la instrucción anterior se convierte en:

```
10 LET r=INT(RND*m)+1
```

si se pone m=6 se obtiene de nuevo el programa del dado.

Es importante tener en cuenta que este facilísimo método de generar situaciones aleatorias, *únicamente* funciona si todas las situaciones tienen las mismas probabilidades.

**Dos programas perfeccionados**

Hasta ahora se ha tratado la aleatoriedad, pero en realidad no se ha escrito ningún programa de juegos completo. La razón de ello es que la aleatoriedad es normalmente una parte de un programa mayor. Aun así, es posible mejorar los programas pequeños examinados anteriormente.

En primer lugar, analicemos el programa de lanzamiento de monedas al aire. Una de las cosas que normalmente se encuentra a faltar en un programa de lanzamiento de monedas es el suspense. Se lanza una moneda... vuela por los aires... da vueltas... será cara... será cruz... finalmente se para. El lanzamiento por ordenador simplemente imprime «cara» o «cruz», antes que uno pueda parpadear. Intentemos retardar la parte del programa que hace la selección para darle un elemento de suspense. Probar con:

```
10 DIM b$(2,5)
20 RAND 0
30 INPUT "¿Quieres jugar? s/n";a$
40 IF a$<>"s" THEN STOP
```

```
50 INK 0: PAPER 6:CLS
60 INPUT "¿cara o cruz?";a$
70 PRINT "O sea que crees que será";a$
80 LET r=INT(RND*15)+10
90 LET b$(1)="cara"
100 LET b$(2)="cruz"
110 LET k=0
120 FOR i=1 TO r
130 LET k=NOT k
140 PRINT AT 5,0;b$(k+1)
150 FOR j=1 TO i
160 NEXT j
170 NEXT i
180 IF a$(1)=b$(k+1,1) THEN PRINT "Has ganado"
190 IF a$(1)<>b$(k+1,1) THEN PRINT "¡¡Has perdido!!"
200 GOTO 30
```

El programa funciona según unos principios diferentes de los del programa de lanzamiento de monedas al aire. Las lineas 10, 90 y 100 seleccionan una cadena matricial que contiene las palabras «cara» y «cruz». El bucle FOR que comienza en la línea 120 y acaba en la línea 170 imprime una de esas dos palabras cada vez que es ejecutado. La instrucción de la línea 130 puede sorprender a algunos lectores; NOT k simplemente convierte k en 0 si es 1, y en 1 si es 0. Es esta «alternancia» de k, cada vez que es ejecutado el bucle, la causante de que salga impreso alternativamente «cara» o «cruz». Si k=0 la línea 140 imprime «cara» y si k=1 imprime «cruz». La línea 80 introduce el elemento aleatorio; r es el número de veces que se ejecuta el bucle. Evidentemente, si r es impar el resultado final será «cara» y si es par el resultado final será «cruz». El toque final de suspense se añade haciendo que las palabras «cara» y «cruz» se alternen cada vez más lentamente a medida que avanza el tiempo, mediante la inclusión de un bucle de retardo en las líneas 150 y 160.

Este programa no es fácil de entender, pero no debe preocupar demasiado si no se puede seguir paso a paso. Mas ade-

lante se describirán con más detalle algunas de las ordenes y las técnicas empleadas.

El segundo programa perfeccionado es un programa de dados. La mejora es obvia: para cada resultado sale impresa la figura de puntos de un dado normal. Se puede conseguir un ahorro en la programación teniendo en cuenta que la figura de tres puntos es igual a la figura de dos puntos más la figura de un punto, que la figura de cuatro puntos es igual que dos figuras de dos puntos, y así sucesivamente con la de cinco puntos (cuatro más uno) y seis (cuatro más dos). El programa resultante es:

```
10 RAND 0
20 LET r=INT(RND*6)+1
30 GOSUB r*100
40 INPUT a$
50 IF a$="s" THEN STOP
60 INK 2: PAPER 6:CLS
70 GOTO 20
100 PRINT AT 5,5;"*"
110 RETURN
200 PRINT AT 0,0;"*"
210 PRINT AT 10,10;"*"
220 RETURN
300 GOSUB 100
310 GOTO 200
400 PRINT AT 0,10;"*"
410 PRINT AT 10,0;"*"
420 GOTO 200
500 GOSUB 400
510 GOTO 100
600 PRINT AT 5,0;"*"
610 PRINT AT 5,10;"*"
620 GOTO 400
```

```
*         *

     *

*         *
```

La única parte ingeniosa del programa es la línea 30, la cual selecciona la subrutina 100 si r es 1, la subrutina 200 si r es 2, etc. Para ejecutar el programa, pulsar ENTER para cada lanzamiento del dado; pulsar «s» para terminar.

## El problema de las cartas

Hasta ahora hemos utilizado los números aleatorios para seleccionar qué acción, entre varias, podría suceder. Da la impresión de que podríamos utilizar los mismos métodos para escribir programas de juegos de cartas. Una baraja de cartas está distribuida en cuatro palos, de 13 cartas cada uno. Hay muchas maneras de utilizar un ordenador para sacar una carta. Una de las más fáciles de comprender es: generar dos números aleatorios, uno del 1 al 4 que seleccione el palo, y otro del 1 al 13 que elija la carta. El problema de este método es que si se da una carta, por ejemplo el as de picas, no hay nada que impida que se vuelva a dar de nuevo. Esta forma de dar cartas es lo mismo que dar una carta, anotar su valor y volverla a poner en la baraja; es dar cartas y volverlas a poner. La forma más usual de dar cartas es repartirlas, y esto es dar sin reponer. Se puede programar esta forma de repartir las cartas, pero se tarda mucho tiempo, normalmente demasiado para un programa BASIC.

El segundo problema con las cartas es que cualquiera que juegue a las cartas nos dirá que gran parte de la diversión proviene de descubrir las probabilidades intentando recordar el orden en que fueron repartidas las cartas. Barajar es una forma muy ineficaz de reordenar las cartas de una baraja; si antes de barajar hay una carta determinada a continuación de otra, después de barajar lo más probable es que las cartas sigan en el mismo orden. Saber aprovecharse de este hecho es lo que caracteriza a los buenos jugadores de cartas. Es divertido imaginar la reacción de un buen jugador de cartas enfrentándose a un ordenador; no hay cartas y el reparto aleatorio es demasiado bueno como para permitir hacer asociaciones entre las parejas de cartas que quedan.

La solución para el reparto sin reposición y para el problema de la ineficacia de la forma de barajar radica en la simulación por el ordenador de una baraja de cartas. Por ejemplo, si se selecciona una cadena que contenga 52 símbolos di-

ferentes (uno para cada carta de cada palo) el reparto se podría realizar mediante la impresión de los símbolos uno por uno. La aleatoriedad se podría garantizar simulando de vez en cuando el acto de barajar. Por ejemplo, para un palo de cartas:

```
10 LET a$= "AC 2C 3C 4C 5C 6C 7C 8C 9C 10C SC QC RC"
20 GOSUB 100
30 LET i=1
40 PRINT a$ (i TO i+2)
50 LET i=i+3
60 IF i=13*3+1 THEN STOP
70 GOTO 40
100 FOR i=1 TO 13
110 LET j= INT (RND*13)
120 LET b$=a$ (j*3+1 TO j*3+3)
130 LET a$=a$ (1 TO j*3) + a$ (j*3+4 TO)
140 LET a$=a$+b$
150 NEXT i
160 RETURN
```

La matriz de la línea 10 representa el palo de corazones, desde el as de corazones (AC) hasta el rey de corazones (RC) pasando por la reina de corazones (QC). La subrutina 100 baraja las cartas de una forma sencilla; selecciona una carta al azar y la coloca al final de la baraja, 13 veces. Las líneas 30 a 70 imprimen las cartas una por una. Hay que tener presente que esta forma de barajar es lenta y no muy buena.

```
 9C
10C
 SC
 QC
 RC
 3C
 8C
 7C
 4C
```

```
6C
2C
5C
AC
```

Si se quiere, se puede llamar de nuevo a la subrutina para barajar más concienzudamente; insertar

```
25 GOSUB 100
```

Como se puede ver, la simple operación de barajar requiere una rutina muy complicada. Pero si no se da mucha importancia a la cuestión de barajar, se puede hacer el programa de un juego de cartas sencillo como el de las veintiuna.

### Las veintiuna

```
10 LET t=0
20 LET u=0
30 INK 6: PAPER 1:CLS
40 REM Juegas una mano
50 GOSUB 400
60 LET t=t+c
70 IF t>21 THEN GOTO 300
80 PRINT "Tienes";t
90 PRINT "Te quedas o sigues s/t"
100 INPUT a$
110 IF a$="s" THEN GOTO 200
120 PRINT "La siguiente carta es-";
130 GOTO 40
200 PRINT "El Spectrum intentará superar-"
210 PRINT "tu total de";t
220 LET a$="Spectrum"
230 GOSUB 400
240 LET u=u+c
250 IF u>21 THEN GOTO 300
260 PRINT "El total del Spectrum es";u
```

```
270 PAUSE 100
280 IF u<t THEN GOTO 230
290 PRINT FLASH 1; "El Spectrum gana": GOTO 330
300 IF t>21 THEN PRINT "Te has";
310 IF u>21 THEN PRINT "El Spectrum se ha";
320 PRINT "PASADO"
330 INPUT "Para otra mano pulsar ENTER";a$
340 GOTO 10
400 REM Da una carta
410 LET c=INT (RND*13)+1
420 LET a$=" "+STR$(c)
430 IF c=1 THEN LET a$= "AS"
440 IF c=11 THEN LET a$= "SOTA"
450 IF c=12 THEN LET a$= "REINA"
460 IF c=13 THEN LET a$= "REY"
470 PRINT a$
480 RETURN
```

```
Tienes 2
Te quedas o sigues s/t
La siguiente carta es - 5
Tienes 7
Te quedas o sigues
La siguiente carta es - 3
Tienes 10
Te quedas o sigues s/t
La siguiente carta es -10
Tienes 20
Te quedas o sigues s/t

El Spectrum intentará superar -
tu total de 20
REY
El total del Spectrum es 13
8
El total del Spectrum es 21
El Spectrum gana
```

Esta es una versión simplificada del juego de las veintiuna que ayuda a ver con claridad los principios básicos del programa. Los valores de las cartas son: AS=1, SOTA=11, REINA

(Q)=12 y REY= 13. La línea 410 reparte las cartas sin reemplazarlas y sin ninguna referencia al palo. La línea 420 convierte el número en una cadena, de manera que las cartas pueden ser números, es decir 2, 3, 4, etc., o palabras, es decir AS, REY, etc.

**Probabilidades desiguales - un método perfeccionado**

Se puede aprovechar una característica especial del Spectrum para generar un número correspondiente al intervalo en el que está comprendido un número aleatorio, tanto en el caso de intervalos desiguales como en el de intervalos iguales. Como se vio anteriormente, si se quieren generar cuatro cosas con probabilidad idéntica se puede utilizar:

```
10 LET r=INT(RND*4)+1
```

pero no funcionará para probabilidades desiguales como las del programa meteorológico. Sin embargo:

```
10 LET r=RND
20 LET w=(r>,4) + (r>,7) + (r>,9) +1
30 PRINT w
40 GOTO 10
```

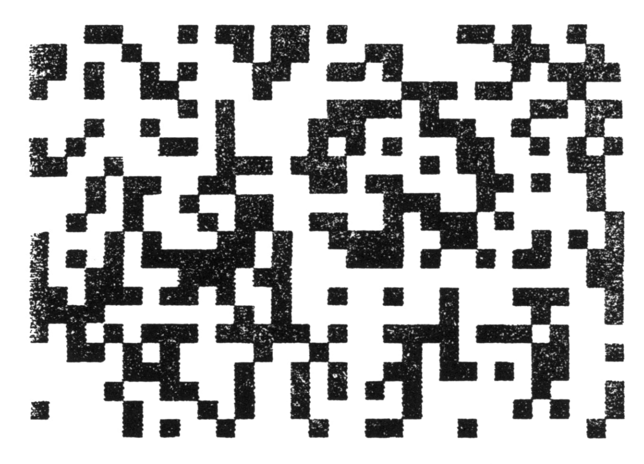
generará números desde el 1 al 4 con las mismas probabilidades (desiguales) que las diferentes condiciones meteorológicas. Funciona porque el Spectrum «resuelve» comparaciones como r<0,4 y dice si se cumple o no (si es cierto o falso); 1 significa cierto y 0 significa falso. Para comprender la línea 20 supongamos que r es 0,5; r es mayor que 0,4 por lo que el primer paréntesis da 1, pero r es menor en todas las otras comparaciones, por lo que el segundo y el tercer paréntesis dan 0. Cuando se suman todos los 1 y 0 con el 1 adicional, se obtiene la respuesta: w es 2. Si se hace la prueba para otros valores de r se llegará a la conclusión de que w es el número del intervalo en que cae r (comenzando en el 1).

Se podría utilizar este método para hacer trampas en los programas de lanzamiento de monedas y de lanzamiento de dados vistos anteriormnte. Pero dejemos esto como un proyecto que cada uno puede hacer por sí mismo.

## La aleatoriedad y la simetría

Sin duda, todo el mundo ha visto las fascinantes imágenes visuales de configuraciones continuamente cambiantes que otros ordenadores generan. El Spectrum también puede hacer este mismo tipo de cosas.Comenzemos con una configuración verdaderamente aleatoria:

```
10 LET x=INT(RND*32)
20 LET y=INT(RND*22)
30 LET c=INT(RND*8)
40 PRINT AT y,x;INK c;"[^8]";
50 GOTO 10
```



(Obsérvese el empleo de la notación de gráficos introducida en la línea 40 del capítulo 2). Las líneas 10 y 20 generan posiciones aleatorias en las que han de salir los caracteres gráficos y la línea 30 seleciona un color de tinta al azar. El resultado de este programa es interesante, pero no es el tipo de

configuración al que se podría mirar durante mucho tiempo. La dificultad radica en que la configuración es demasiado aleatoria. Las configuraciones interesantes y en cambio continuo deben usar la aleatoriedad para variar, pero la deben usar de una forma controlada. Uno de los principios de organización básicos de la naturaleza es la simetría, y este principio se puede aplicar a las configuraciones del ordenador para introducir orden.

En el BASIC del Spectrum es muy fácil manejar la simetría cuádruple, la cual es suficientemente eficaz para producir muchas configuraciones interesantes. Se entenderá mejor el concepto de simetría cuadrúple si imaginamos la pantalla del Spectrum dividida en cuatro partes. Si se imprime un carácter en la primera cuarta parte y luego se consigue que salga impreso también en las otras tres partes se tendrá una configuración simétrica. Si nos imaginamos que las dos líneas que dividen la pantalla en cuatro cuartos son espejos, las posiciones de los otros tres puntos se pueden considerar imágenes reflejadas del original. Si las coordenadas del punto original del primer cuarto son X, Y, las coordenadas de las imágenes reflejadas serán 31-X,Y, X,21-Y y 31-X,21-Y. La mejor manera de comprobar que esto es así consiste en utilizar una hoja de papel gráfico para dibujar la pantalla y calcular las coordenadas. Utilizando estos hechos tan sencillos se puede escribir un programa del calidoscopio.

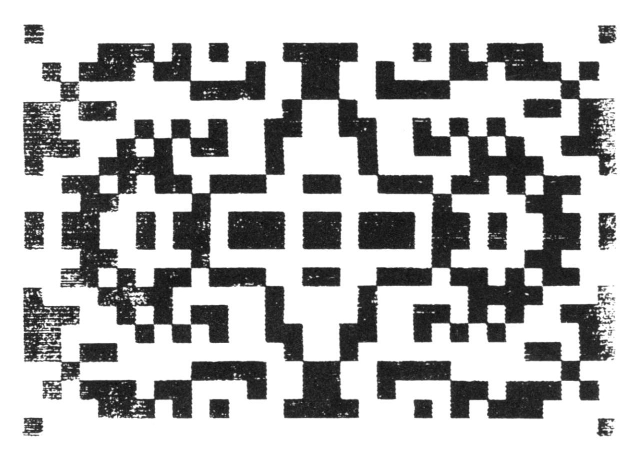
```
10 LET m=31
20 LET n=21
30 LET x=RND*m/2
40 LET y=RND*n/2
50 LET c=INT(RND*8)
60 INK c
70 PRINT AT y,x;"[^8]";
80 PRINT AT y,m-x;"[^8]";
90 PRINT AT n-y,x;"[^8]";
100 PRINT AT n-y,m-x;"[^8]";
110 GOTO 30
```

Las líneas 30 y 40 generan la posición del bloque de gráfico y la línea 50 genera un color de tinta. La línea 60 selecciona el color de tinta para todas las instrucciones PRINT siguientes (líneas 70 a 100).

Aplicando esta idea básica de la simetría cuádruple es posible añadir otras características de control para generar configuraciones más interesantes. Podría ser más interesante que los cambios aleatorios fueran en «ciclos» que comienzan en el medio y continúan a partir de ahí. En el ejemplo siguiente se puede ver esta idea en acción:

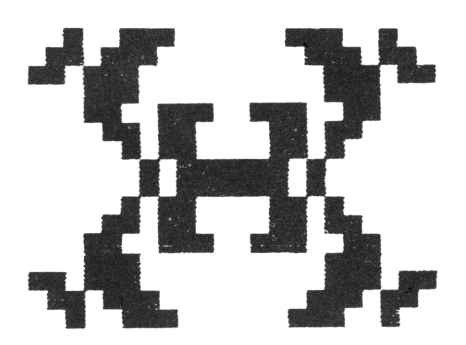
```
10 LET m=31
20 LET n=21
30 FOR x=0 TO m/2
40 LET y=RND*n/2
50 LET c=INT(RND*8): INK c
60 PRINT AT y,x;"[^8]";
70 PRINT AT y,m-x;"[^8]";
80 PRINT AT n-y,x;"[^8]";
90 PRINT AT n-y,m-x;"[^8]";
100 NEXT x
110 GOTO 30
```

Es difícil capturar la imagen resultante de este programa pues tiene movimiento, pero un resultado típico (que sería en color) podría tener este aspecto:



Se puede utilizar la simetría y la aleatoriedad para generar configuraciones diferentes a las usuales. Si partimos de un punto situado en el centro de la pantalla y le hacemos andar aleatoriamente por la pantalla sumando −1,0 o 1 a cada una de sus coordenadas se obtendrá una línea aleatoria muy interesante. Pero si se utiliza también la rutina de la simetría cuádruple para reflejar la línea en los otros cuartos, el resultado es un conjunto de formas fascinantes:

```
10 LET m=31
20 LET n=21
30 LET x=m/2
40 LET y=n/2
50 GOSUB 100
60 LET x=x+RND*2-1
70 LET y=y+RND*2-1
80 GOTO 50
100 PRINT AT y,x;"[^8]";
110 PRINT AT y,m-x;"[^8]";
120 PRINT AT n-y,x;"[^8]";
130 PRINT AT n-y,m-x;"[^8]";
140 RETURN
```



Podríamos continuar mucho tiempo con el tema de las configuraciones aleatorias pero dejaremos el resto del tema para que cada uno lo explore por sí mismo.

# 4. GRAFICOS DE ALTA RESOLUCION

En el Capítulo 2 se presentó la idea en que se basan los gráficos del Spectrum y se describieron los dos tipos de puntos, papel y tinta, que componen la imagen. Imprimiendo caracteres y formas gráficas en las posiciones de carácter de la pantalla, es posible crear gráficos impresionantes y de gran definición, pero si se quieren dibujar gráficos o trazar figuras es preciso poder cambiar los puntos de la pantalla de una forma que no tenga en cuenta las posiciones de carácter: entramos en el reino de los gráficos de alta resolución. La orden PLOT puede convertir un punto cualquiera de papel a tinta o viceversa sin afectar a los otros, y la orden DRAW puede generar líneas rectas y curvas en cualquier posición de la pantalla. Tal vez la orden más avanzada de los gráficos de alta resolución es CIRCLE que, como su nombre indica, sirve para dibujar un círculo de un determinado radio. Para un ordenador de un precio razonable como el Spectrum, esta capacidad de alta resolución resulta impresionante. De todas formas, si se quieren generar gráficos de alta resolución a todo color, surgirán problemas.

## Especificación de un punto

Obviamente, si se van a ejecutar órdenes que alteren puntos individuales en la pantalla, se va a necesitar alguna forma de especificar qué puntos. Esto se soluciona fácilmente mediante la estratagema usual de decir el número de la fila y de la columna en cuestión. Normalmente, el número de la columna recibe el nombre de «abscisa» o «coordenada x» y el número de la fila recibe el nombre de «ordenada» o «coordenada y». Todos los números de fila y de columna comienzan desde cero y, como la pantalla del Spectrum es un rectángulo compuesto por 256 puntos horizontales y 176 puntos verticales, esto significa que la abscisa va del 0 a 255 y la ordenada del 0 al 175. La única complicación adicional es que la ordenada comienza en el 0 en la parte «*inferior*» de la pantalla. Es justamente lo contrario al número de línea de una instrucción PRINT AT. Dicho de otra forma, en los gráficos de alta resolución el punto de abscisa 0 y ordenada 0 está en la esquina inferior izquierda.

## Las órdenes de gráficos de alta resolución

Antes de continuar estudiando la forma de utilizar los gráficos de alta resolución en programas, merece la pena mirar detalladamente las tres órdenes de alta resolución. La orden de alta resolución básica es:

```
PLOT x,y
```

que cambia el punto x,y en un punto de tinta. La x y la y pueden ser expresiones aritméticas. En realidad la única restricción es que x e y deben ser coordenadas válidas de la pantalla. Dicho de otra forma, x tiene que ser una abscisa comprendida entre 0 y 255 y tiene que ser una ordenada comprendida entre 0 y 175; cualquier otro valor produce la impresión de un mensaje de error. Para ver en acción la orden PLOT, ejecutar:

```
10 INPUT x,y
20 PLOT x,y
30 GOTO 10
```

Para llenar la pantalla de puntos de tinta ejecutar.

```
10 FOR x=0 TO 255
20 FOR y=0 TO 175
30 PLOT x,y
40 NEXT y
50 NEXT x
```

No hay que olvidar que, por ahora, únicamente podemos utilizar órdenes PLOT para generar puntos de tinta. Una vez se hayan presentado todas las órdenes de gráficos de alta resolución se tratará la forma de generar puntos de papel.

La orden DRAW se utiliza conjuntamente con la orden PLOT para generar líneas. La instrucción

```
DRAW x,y
```

dibujará una línea que empieza en el lugar donde se trazó el último punto. El último punto de la línea está a x puntos de la abscisa y a y puntos de la ordenada del principio. El hecho de que el punto inicial de la línea dependa de una orden previa, y que no se especifique explícitamente el punto final, producen la sensación de que la orden DRAW es difícil de utilizar. Sin embargo, en la práctica encaja muy bien con la forma normal de utilizar los gráficos de alta resolución. La orden combinada

```
PLOT x,y:DRAW xd,yd
```

dibuja una línea comprendida entre el punto x,y y el punto x+xd,y+yd. Un poco de aritmética nos mostrará que

```
PLOT x1,y1:DRAW x2-x1,y2-y1
```

dibujará una línea compendida entre x1,y1 y x2,y2. Para investigar un poco más la orden DRAW ejecutar:

```
10 PLOT 100,50
20 DRAW 25,25
```

Esta orden situa un punto en 100,50 y dibuja una línea diagonal «hacia la parte superior derecha». Pero

```
10 PLOT 100,50
20 DRAW -25,-25
```

también dibuja una línea diagonal empezando desde el mismo punto pero ahora «hacia la parte inferior izquierda». Este último ejemplo muestra una diferencia importante entre PLOT y DRAW: Si bien los números negativos carecen de sentido cuando se utilizan con la orden PLOT, son necesarios si se quiere dibujar en la pantalla líneas que se extiendan desde la parte superior a la inferior o de la derecha a la izquierda.

Uno se puede preguntar qué sucede si a una orden DRAW le sigue inmediatamente otra. ¿Dónde comienza la segunda línea? La respuesta es que la línea comienza en el punto final de la primera orden DRAW. Una forma eficaz de verlo es imaginar un «cursor gráfico» invisible que se mueve por la pantalla a medida que se van trazando puntos. En consecuencia, después de una orden el cursor se sitúa en la posición del último punto trazado. La orden DRAW comienza siempre dibujando una línea desde la posición actual del cursor. Se pueden utilizar numerosas órdenes DRAW, una después de otra, para generar contornos continuos. Por ejemplo, el siguiente programa dibuja un cuadro:

```
10 PLOT 50,100
20 DRAW 50,0
30 DRAW 0,-50
40 DRAW -50,0
50 DRAW 0,50
```

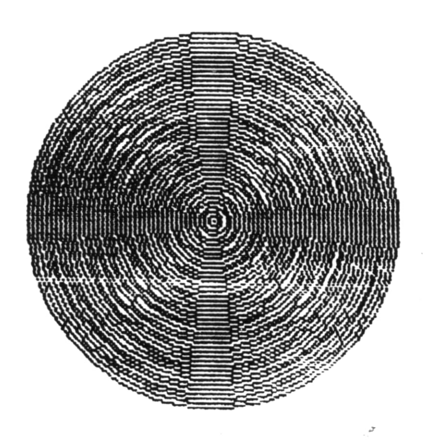
Podemos considerar que la primera orden desplaza el cursor gráfico a 50,100. Las órdenes DRAW generan los lados del cuadrado y desplazan el cursor para la siguiente orden DRAW.

La última orden de alta resolución es

```
CIRCLE x,y,r
```

que trazará lo más parecido a un círculo que el Spectrum puede dibujar en su rejilla de 256 por 176 puntos. El círculo tendrá su centro en el punto x,y, y tendrá un radio determinado por r. Para una demostración de la orden CIRCLE, ejecutar:

```
10 FOR r=0 TO 80 STEP 2
20 CIRCLE 128,80,r
30 NEXT r
```



Este programa genera una configuración interesante pues traza numerosos círculos todos centrados en 120,80. La orden CIRCLE es relativamente fácil de utilizar; en realidad, la única cosa que hay que vigilar es no salirse de la pantalla. Mas adelante volveremos a tratar el tema de los círculos.

*INVERSE,OVER y atributos*

Las órdenes de gráficos de alta resolución pueden ser utilizadas conjuntamente con las órdenes INVERSE y OVER y con las órdenes de los atributos INK,PAPER,FLASH y BRIGHT. No es difícil calcular los efectos que producen esas órdenes mientras se tenga presente que los atributos sólo se pueden asignar a posiciones de carácter enteras y no a puntos individuales. Incluso cabe la posibilidad de poner estas órdenes dentro de órdenes de gráficos de alta resolución, de la misma forma que se las puede poner en las instrucciones PRINT. Sin embargo, sus efectos pueden ser bastantes extraños, tal como veremos. Es más fácil analizar las dos órdenes INVERSE y OVER antes de analizar las órdenes de atributos.

Si se traza un punto después de la orden INVERSE 1, en vez de un punto de tinta se generará un punto de papel. Esto sucede con todas las órdenes gráficas, luego si se acaba de generar un círculo de puntos de tinta mediante la orden CIRCLE, se puede cambiar, es decir, convertirle otra vez a puntos de papel, con

```
CIRCLE INVERSE 1,x,y,r
```

Dicho de otra manera, podemos considerar que PLOT INVERSE 1,x,y «destraza» o borra un punto. Una cosa a la que hay que prestar atención cuando se utiliza INVERSE para cambiar líneas que han sido trazadas por DRAW, es especificar exactamente la misma línea que la vez anterior. Por ejemplo, si se genera una línea mediante

```
10 PLOT 10,10:DRAW 100,100
```

Hay que utilizar

```
10 PLOT 10,10:DRAW INVERSE 1,100,100
```

y no

```
10 PLOT 100,100;DRAW INVERSE 1,-100,-100
```

que intenta borrar la línea comenzando en el punto final de las líneas originales. Para estar seguros de borrar todos los puntos hay que destrazar siempre en la misma dirección.

También se puede utilizar la orden OVER con gráficos de alta resolución, pero como normalmente las órdenes de alta resolución tan sólo generan puntos de tinta, su acción es relativamente sencilla: Si ya hay un punto de tinta presente en la posición en que una orden de gráficos de alta resolución pretende generar un punto de tinta, se obtendrá al final un punto de papel; en cambio, si hay un punto de papel presente, una orden de gráficos de alta resolución generará un punto de tinta. Dicho de otra forma, después de una orden OVER 1, las órdenes de gráficos de alta resolución convierten los puntos de papel existentes en puntos de tinta y viceversa.

Si se combina OVER 1 con INVERSE 1, los dos efectos quedan anulados. INVERSE 1 hace que todas las órdenes de alta resolución generen puntos de papel, pero debido a las características de OVER 1 (ver Capítulo 2), un punto de papel trazado en la misma posición que un punto de tinta existente genera un punto de tinta. Puesto que un punto de papel trazado en la misma posición que un punto de papel existente genera un punto de papel, está claro que no se produce cambio alguno. Esto no significa que

```
PLOT inverse 1,OVER 1,x,y
```

sea inútil aunque no altere ninguno de los puntos de la pantalla; puede servir para mover el cursor de gráficos sin tener que trazar nada.

El tema de los atributos y de los gráficos de alta resolución es un poco más complicado que INVERSE y OVER, ya que

los atributos están asignados a posiciones de carácter enteras y no a puntos individuales. Por lo tanto, aunque se pueden utilizar las órdenes INK, PAPER, BRIGHT y FLASH con las órdenes de alta resolución e incluso se las puede incluir dentro de dichas órdenes, afectan a toda la posición de carácter donde está incluido un punto. En resumen: si se traza un punto de tinta verde, todos los demás puntos de tinta de esa posición de carácter pasarán a ser de color verde. Lo mismo pasa al trazar puntos de papel, puntos intermitentes y puntos brillantes. Dicho de otra forma, cualquier atributo que afecte a un punto individual es inmediatamente compartido por todos los otros puntos de la misma posición del carácter. Para verlo en acción, ejecutar:

```
10 PLOT INK, 3,10,10
20 DRAW INK 3,100,100
30 PLOT INK 5,10,100
40 DRAW INK 5, PAPER 2,100,-100
```

Las dos instrucciones iniciales dibujan una línea diagonal roja. Las líneas 30 y 40 dibujan una línea diagonal que cruza a la primera. Hay dos cosas a tener en cuenta sobre los resultados de este programa. En primer lugar, donde las dos líneas pasan por una misma posición de carácter sólo puede haber UN color de tinta, y es el color de la última línea que se trazó, es decir tinta 5, cian. En segundo lugar, lo que es más sorprendente, la orden PAPER de la línea 40 afecta a todas las posiciones de carácter que la línea atraviesa, a pesar de que la orden DRAW genera únicamente puntos de tinta. Lo dicho se debe considerar una advertencia para utilizar con cuidado los atributos en las órdenes de alta resolución.

En términos prácticos, esta restricción que afecta a los atributos y a los colores en particular significa que es mucho más fácil trabajar con gráficos de alta resolución de sólo dos colores. Si se quiere utilizar toda la gama de colores del Spectrum se deberá planificar las imágenes para que no se encuentren más de dos colores en una posición de carácter. Esto

resulta imposible muchas veces, a menos que se esté tratando con formas muy sencillas. Sin embargo, la utilización de solo dos colores en gráficos de alta resolución no es insuficiente para muchas de las tareas a las que son aplicables los gráficos de alta resolución; por ejemplo, trazar gráficos.

## Círculos y Elipses

Aunque el Spectrum dispone de una orden para generar círculos de gran calidad a alta velocidad, merece la pena saber generar círculos utilizando los principios básicos. Además, una vez se sepa dibujar un círculo, pronto se descubrirá la forma de dibujar una elipse; si no se sabe para qué puede servir dibujar un elipse, véase más adelante el programa de la «lata de judías cocidas».

Las coordenadas de cualquier punto de un círculo que tenga el centro en x1, y1 y de un radio r satisfacen las dos ecuaciones siguientes:

```
x=r*SIN(t)+x1
```

y

```
y=r*COS(t)+y1
```

para cualquier valor de t. No importa que no se entiendan estas ecuaciones; se las puede utilizar de todas formas. Lo importante es que si se da un valor a t y se calculan las dos ecuaciones se obtendrá un punto del círculo. Hacer la prueba con el siguiente programa:

```
10 LET r=60
20 LET x1=100
30 LET y1=60
40 LET t=RND*6.283
50 PLOT r*SIN(t)+x1,r*COS(t)+y1
60 GOTO 40
```

Veremos aparecer en la pantalla un círculo en un orden aleatorio. Hemos utilizado la orden RND para generar valores aleatorios de t y por lo tanto puntos aleatorios del círculo. Si se quiere, se puede comenzar con t=0 y, aumentando t lentamente, trazar todos los puntos del círculo. Sucede justamente que el círculo se cierra cuanto t alcanza el extraño valor de 6,283, que resulta ser el doble del valor de π. (Hay una razón matemática muy importante para ello, pero no nos debe preocupar). Podemos utilizar la tecla PI del Spectrum en lugar de una aproximación:

```
10 LET r=60
20 FOR t=0 TO 2*PI STEP .01
30 PLOT r*SIN t + 100,r*COS t +80
40 NEXT t
```

Si se ejecuta el programa anterior saldrá un círculo, pero mucho más lentamente que uno generado por la orden CIRCLE. Probar con valores diferentes del paso para generar círculos de diferentes «redondeces».

Los círculos generados utilizando SIN y COS deberían, al menos en teoría, ser tan precisos como permite la definición limitada de la pantalla de 256 por 176 del Spectrum. El principio en que se basa la orden CIRCLE para dibujar círculos no se encuentra en ningún sitio del manual del Spectrum y no hay ninguna garantía de que los círculos sean perfectos. Resulta interesante comparar los dos métodos. El siguiente programa dibuja primero un círculo utiliando la fórmula SIN/COS y luego dibuja el mismo círculo utilizando la orden CIRCLE. Los puntos donde coinciden los dos círculos se convierten en puntos de papel debido a la orden OVER 1 de la línea 50. De este modo, los únicos puntos de tinta que quedan al final del programa son los puntos donde no coinciden los dos círculos generados.

```
10 LET r=60
20 FOR t=0 TO 2*PI STEP .01
```

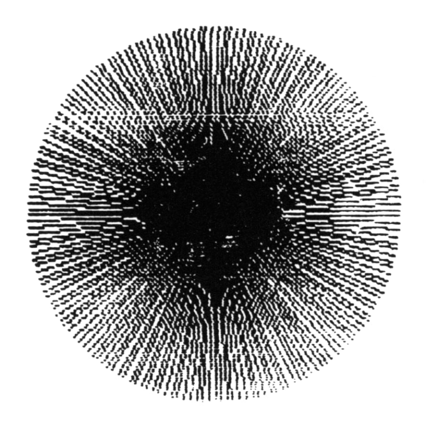
```
30 PLOT r*SIN t + 100,r*COS t +80
40 NEXT t
50 OVER 1
60 CIRCLE 100,80,60
```



Aunque los dos métodos generan círculos ligeramente diferentes la diferencia es mínima, y si se considera la velocidad de la orden CIRCLE la inexactitud es disculpable.

El conocer las coordenadas de los puntos comprendidos en un círculo tiene otras aplicaciones además de la de dibujar círculos. Por ejemplo, si se dibujan líneas desde el centro a la circunferencia se pueden conseguir algunas configuraciones interesantes. Ejecutar:

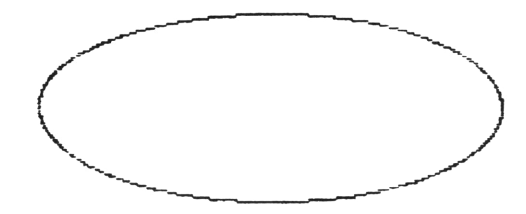
```
10 LET r=80
20 LET c=INT(RND*8)
30 INK c
40 PAPER 9
50 CLS
60 FOR t=0 TO 2*PI STEP (RND+RND)/10
70 LET x=r*SIN t
80 LET y=r*COS t
90 PLOT 128,80
100 DRAW x,y
110 NEXT t
120 PAUSE 100
130 GOTO 20
```



La línea 90 mueve el cursor gráfico al centro del círculo y la línea 100 dibuja una línea a la circunferencia. Cabe señalar que se ha utilizado el color 9 para generar un fondo que contraste con la configuración.

Una vez que se conoce la forma de generar círculos, no es difícil ampliarla para generar elipses. Probablemente todo el mundo sabe ya lo que es una elipse, un círculo de forma «aplanada». Es lo que se ve cuando se mira un círculo en un ángulo oblicuo, y resulta muy eficaz cuando se quiere generar cualquier dibujo tridimensional de objetos circulares. Para dibujar una elipse se puede utilizar la misma fórmula que para dibujar un círculo pero con dos valores para el radio; un radio horizontal y un radio vertical. Hacer la prueba con el siguiente programa:

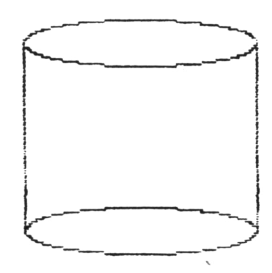
```
10 LET r1=100
20 LET r2=40
30 FOR t=0 TO 2*PI STEP .01
40 PLOT r1*SIN t + 128,r2* COS t +80
50 NEXT t
```



Dibuja una elipse con un radio horizontal de 100 y un radio vertical de 40. Intentarlo con otros valores de r1 y r2 para generar elipses de formas diferentes.

A título de ejemplo de la aplicación de las elipses para dibujar un objeto tridimensional, ejecutar el siguiente programa «lata»:

```
10 FOR t=0 TO 2*PI STEP .02
20 PLOT 50*SIN t+128,10*COS t+100
25 PLOT 50*SIN t+128,10*COS t+20
30 NEXT t
40 PLOT 128+50,20
50 DRAW 0,80
60 PLOT 128-50,20
70 DRAW 0,80
```



Se debe poder distinguir las dos elipses de este programa.

## El juego de las flechas

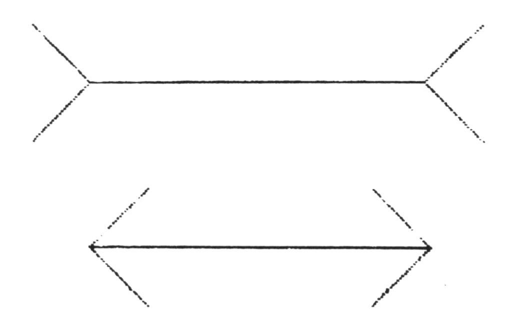
El juego de las flechas es una aplicación simple de los gráficos de alta resolución. Se dibujan dos flechas de una longitud aleatoria; la longitud de la segunda flecha se puede modificar pulsando la tecla «flecha izquierda», que la reduce, o pulsando la tecla «flecha derecha», que la alarga. El objetivo del juego es conseguir que la segunda flecha tenga la misma longitud que la primera. Parece muy fácil hasta que se intenta, pero hay una conocida ilusión óptica en acción.

```
10 LET L=INT((RND*20)+140)
20 PLOT 50,120
30 DRAW L,0
40 PLOT 50,120
50 DRAW -25,25
60 PLOT 50,120
70 DRAW -25,-25
80 PLOT 50+L,120
90 DRAW 25,25
100 PLOT 50+L,120
110 DRAW 25,-25
120 LET s=INT((RND*50)+100)
130 GOSUB 500
140 LET a$=INKEY$
150 IF a$="" THEN GOTO 140
160 IF a$="s" THEN GOTO 300
170 IF a$="5" THEN GOSUB 500: LET s=s-1: GOTO 130
180 IF a$="8" THEN GOSUB 500: LET s=s+1: GOTO 130
190 GOTO 140

300 PRINT AT 18,0;"Flecha superior=";L
310 PRINT AT 19,0;"inferior=";s
320 PRINT AT 20,0;"Hay una diferencia de"
330 PLOT 50,4
340 DRAW ABS(L-s),0
350 IF L=s THEN PLOT OVER 1;50,4
360 STOP

500 OVER 1
510 PLOT 50,50
520 DRAW s,0
530 PLOT 50,50
540 DRAW 25,25
550 PLOT 50,50
560 DRAW 25,-25
570 PLOT 50+s,50
580 DRAW -25,-25
590 PLOT 50+s,50
```

```
600 DRAW -25,25
610 OVER 0
620 RETURN
```



Las líneas 10 a 110 dibujan en la pantalla la primera flecha. La línea 120 selecciona una longitud aleatoria para la segunda flecha, que es dibujada por la subrutina 500. (Observar la utilización de OVER en la subrutina 500). La primera llamada a la subrutina 500 genera la flecha en la pantalla y una segunda llamada, con la misma longitud, la borra de la pantalla. Las líneas 140 a 180 comprueban cuál es la tecla pulsada, en el caso de que se esté pulsando alguna. La tecla de la flecha a la derecha aumenta la longitud de la línea y vuelve a trazar la flecha, y la tecla de la flecha a la izquierda reduce la longitud de la línea y la vuelve a trazar. Si se pulsa la tecla s, las líneas 300 a 350 imprimen la longitud de las flechas y dibujan la diferencia en forma de una línea corta.

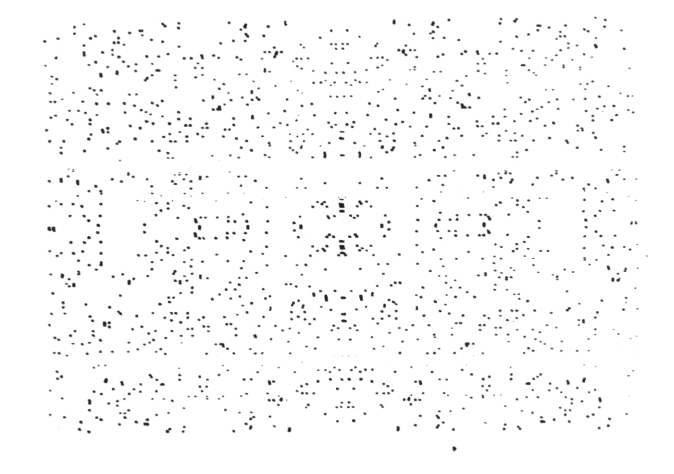
Aunque es un programa bastante simple, cuando se ejecuta da la impresión de que la segunda flecha se mueve en respuesta a la tecla que se esté pulsando.

**Demasiada resolución**

Un error típico de principiantes es la utilización excesiva de gráficos de alta resolución. Tal como se ha dicho anterior-

mente, hay cosas que son más fáciles y mejores de hacer con gráficos de baja resolución. A título de ejemplo, pensemos en los programas de simetría y aleatoriedad analizados en el Capítulo 2. Si se piensa escribir un programa de este tipo directamente, se puede caer en la tentación de utilizar gráficos de alta resolución para obtener una configuración más complicada. El primer resultado de este enfoque es que la configuración puede ser únicamente de dos colores, mientras que la versión de baja resolución del programa puede utilizar los ocho colores. Incluso si se acepta esta restricción de utilizar solo dos colores y si continúa con el programa de alta resolución, el efecto global puede ser decepcionante. Hacer la prueba con:

```
10 OVER 1
20 LET x=INT(RND*128)
30 LET y=INT(RND*88)
40 PLOT x,y
50 PLOT 255-x,y
60 PLOT x,175-y
70 PLOT 255-x,175-y
80 GOTO 20
```



Si bien es cierto que este programa genera una configuración compleja, para mucha gente es demasiado complicada como para ser interesante; los puntos son demasiado pequeños y la resolución es demasiado alta.

## Una forma avanzada de DRAW

Este apartado analiza una forma ligeramente más avanzada de la orden DRAW; se puede omitir en una primera lectura. La forma completa de la orden DRAW es

```
DRAW x,y,t
```

donde x e y tienen el mismo sentido que antes, y t es un ángulo medido en radianes. La mejor manera de explicar qué hace la nueva forma de DRAW es poner un ejemplo:

```
10 PLOT x1,y1:DRAW x2,-x1,y2-y1
```

Dibuja una línea recta entre los puntos x1,y1 y x2,y2. La orden,

```
10 PLOT x1,y1:DRAW x2-x1,y2-y1,t
```

también dibuja una línea entre x1,y1 y x2,y2 pero en lugar de una línea recta traza una parte de un círculo. El tamaño de esa parte depende del valor de t. Si no se está familiarizado con la medición de ángulos en radianes resultará útil la siguiente tabla:

| *t* | *parte de un círculo* |
|---|---|
| 2*PI | entero |
| PI | mitad |
| PI/2 | un cuarto |

El problema con la versión ampliada de DRAW consiste en que es difícil garantizar que la curva dibujada permanezca

en la pantalla en toda su longitud. No obstante, cuando se necesita dibujar parte de un círculo DRAW es la mejor solución. Por ejemplo, para dibujar un semicírculo:

```
10 PLOT 100,80
20 DRAW 50,50,PI/2
```

Hay dos formas de dibujar un arco de circunferencia entre dos puntos: en el sentido de las agujas del reloj o en el sentido contrario. Para ver la forma de dibujar el otro arco en el ejemplo anterior, cambiar PI/2 por −PI/2.

La orden DRAW ampliada no se utiliza frecuentemente, simplemente porque dibujar partes de círculos no es algo que surja muy a menudo.

# 5. EL SONIDO

La posibilidad de generar sonidos a través de un diminuto altavoz interno es una característica del Spectrum que se puede utilizar para dar una nueva emoción a los juegos. Además, el que siempre se haya sentido atraído por la música pero no haya tenido la paciencia, el tiempo o la capacidad necesarias para aprender a tocar un instrumento musical, encontrará en el Spectrum un medio idóneo para descubrir ese mundo. No obstante, hay numerosos problemas con el sonido en el Spectrum: únicamente puede producir una nota cada vez, no es posible controlar ni el volumen ni la calidad de una nota, los efectos sonoros son difíciles de obtener desde el lenguaje BASIC, y, por último, a diferencia de otros ordenadores más caros, el Spectrum sólo puede hacer una cosa cada vez; o genera ruidos o ejecuta programas. Como resultado de este último problema, si se quiere ejecutar un juego con acompañamiento musical será difícil conseguirlo desde el BASIC. Es posible superar algunos de estos problemas mediante una programación inteligente, pero si después de leer este capítulo alguien se siente totalmente enganchado en el tema la música

de ordenador, tendrá que pasar al campo más difícil de la escritura de programas en código máquina, o mejorar su Spectrum. No obstante, antes de tomar medidas tan drásticas, hay mucho que aprender y muchos momentos divertidos que pasar con los sonidos del Spectrum, utilizando simplemente el BASIC.

## *BEEP y PAUSE*

La única orden de sonido que el Spectrum tiene es BEEP:

```
BEEP duración,tono
```

que genera en el altavoz un sonido de una duración determinada y con un tono especificado. La «duración» puede ser cualquier expresión válida y se mide en segundos. El «tono» también puede ser cualquier expresión válida, pero se mide de una forma ligeramente más complicada, como semitonos por arriba y por debajo de la C media (más adelante, se explicará detalladamente). Lo más importante a tener en cuenta sobre la orden BEEP es que tarda «duración» segundos en concluir. Esto significa que si se genera un sonido durante diez segundos, por ejemplo, no se puede esperar que el Spectrum haga ninguna otra cosa durante ese tiempo. Esta característica hace que BEEP se parezca más a una PAUSE que a un sonido. Como contraste, algunos ordenadores proporcionan órdenes de sonido que inician la salida de un sonido y casi inmediatamente pasan a ejecutar la siguiente instrucción del programa. Obviamente esta es una situación mucho más interesante; más adelante investigaremos la forma de obtener un efecto similar con el Spectrum.

La orden BEEP proporciona una forma de generar el sonido de una nota, durante un determinado espacio de tiempo, pero en muchas aplicaciones, por ejemplo al interpretar música, también resulta muy importante la capacidad de producir un silencio de una duración determinada. Esta característica

es fácilmente obtenible en el Spectrum mediante la utilización de la orden PAUSE:

```
PAUSE duración
```

Esta orden simplemente hace que el ordenador se quede a la espera sin hacer absolutamente nada, por un determinado período de tiempo. No obstante, hay dos cosas con las que hay que tener cuidado cuando se utilizan conjuntamente las órdenes PAUSE y BEEP. La primera es simplemente que la orden PAUSE «duración» se mide en cincuentavos de segundo (sesentavos de segundo en Estados Unidos y en algunos otros sitios) pues depende de la velocidad de presentación de cuadros en la pantalla de televisión; para más detalles consultar el Capítulo 8. La solución más fácil a este problema es escribir todas las órdenes PAUSE así:

```
PAUSE duración*50
```

Convierte automáticamente los segundos de la «duración» en cincuentavos de segundo. El segundo problema radica en que la duración de una orden PAUSE puede verse reducida si se pulsa cualquier tecla del teclado, sin que exista ninguna otra solución que ser conscientes de ello y mantener los dedos lejos del teclado mientras haya música con pausas.

En realidad el método que utiliza el Spectrum para generar sonido es muy simple. El sonido es una vibración del aire y el Spectrum genera estas vibraciones moviendo hacia dentro y hacia fuera el cono de su diminuto altavoz. La frecuencia del sonido producido depende del número de veces por segundo que se mueve el cono del altavoz; de hecho es el único control que podemos ejercer sobre el sonido del Spectrum. El cono únicamente puede estar en dos estados: completamente dentro o completamente fuera. Un movimiento de dentro a fuera genera un chasquido. Si se quiere escuchar este chasquido ejecutar:

```
BEEP .001,0
```

Si se repite este movimiento se comenzará a escuchar un sonido continuo en lugar de un solo chasquido, y a medida que aumenta el número de chasquidos por segundo aumenta el tono o la frecuencia de la nota. Si se quiere escuchar, ejecutar:

```
10 FOR i=-60 TO 69
20 BEEP .5,i
30 NEXT i
```

Este ejemplo pone de manifiesto también la escala de notas que se puede generar. En la práctica, la escala de notas útiles es más pequeña porque las notas muy altas suenan desiguales y las notas bajas suenan ásperas rápidamente.

**Interpretación de música escrita**

Interpretar un tono es sencillo. Basta generar el correcto tono en el tiempo correcto y con la duración correcta. El único problema es que la música es mucho más antigua que los ordenadores digitales, y los músicos han creado su propia escritura: la notación musical. Afortunadamente, la notación musical no es demasiado difícil de entender, y cualquier persona puede aprender a leer música en un corto espacio de tiempo.

La notación musical nos expresa dos informaciones: qué tono debe tener un sonido y cuánto debe durar. El tono de una nota lo indica su posición en una fila de líneas llamada pentagrama. Cada nota recibe un nombre según la línea en que está situada, tal como se muestra en la figura 1.
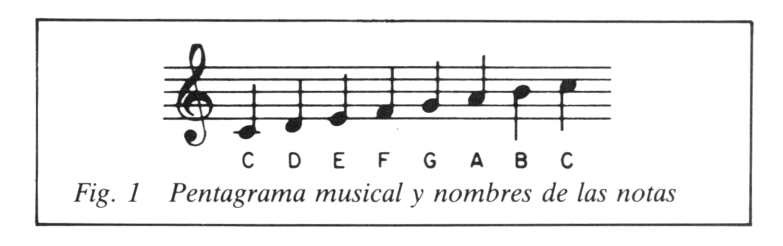



*Fig. 1 Pentagrama musical y nombres de las notas*

Comenzando desde C media, los nombres de las notas van alfabéticamente hasta G, y comienzan de nuevo en A, luego B, y vuelta a C. No es un accidente que C aparezca más de una vez; se debe al hecho de que en realidad las notas con el mismo nombre suenan igual. La distancia musical entre dos notas con el mismo nombre recibe el nombre de «octava».

El único problema consiste en convertir las notas en valores de «tonos» adecuados para el Spectrum. La línea marcada con la C media corresponde a un valor de tono cero; aumentando el valor del tono en uno se sube un «semitono», y disminuyendo el valor del tono en uno se baja un «semitono». Si se comienza en C media con un valor de tono de cero y se quiere subir una nota entera a D, es correcto sumarle 2. Si 1 corresponde a un semitono, 2 corresponde a un tono completo. El siguiente paso consiste en ir de D a E sumando 2 al valor del tono, lo cual también es correcto, pero sumar 2 a E no nos lleva a la nota F. En realidad el problema radica en que, al menos en la música occidental, no siempre hay un tono entero entre dos notas determinadas. Esto se puede observar también examinando el teclado de un piano estándar. La plantilla de teclas blancas y negras nos indica a qué «distancia» se encuentra cada par de notas comenzando en media C; la secuencia es

```
  T  T  S  T  T  T  S
C  D  E  F  G  A  B  C
```

De forma que ir de E a F únicamente requiere aumentar en 1 al valor del tono del Spectrum.

Si todo esto parece un poco enbrollado y uno piensa que en realidad todas las notas musicales tienen entre sí la misma distancia de «semitonos» probar los siguientes programas:

```
10 FOR i=0 TO 14 STEP 2
20 BEEP .5,i
30 NEXT i
```

y

```
10 DATA 0,2,4,5,7,9,11,12
20 FOR i=1 TO 8
30 READ p
40 BEEP .5,p
50 NEXT i
```

El primer programa interpreta una «escala» suponiendo que hay un tono entero entre cada nota, y el segundo programa interpreta la escala (correcta) de C. Se debería preferir la escala generada por el segundo programa, pero incluso aunque alguien no esté de acuerdo, ésta es la que la mayoría de la gente prefiere. Además, el segundo programa introduce la idea de una instrucción de datos que contiene los valores de los tonos que tiene que interpretar el Spectrum, una idea que utilizaremos repetidamente. Al utilizar esta plantilla de «distancias» tono/semitono se llega a la siguiente correspondencia entre líneas del pentagrama y valores de tonos.
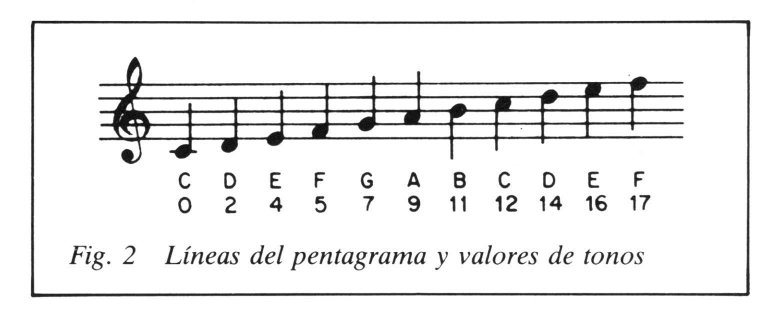



*Fig. 2 Líneas del pentagrama y valores de tonos*

Lo único que hay que hacer para convertir una escritura musical en valores de tonos es utilizar la correspondencia de la fig. 2. La única complicación adicional es que algunas veces los músicos quieren hacer el tono de una nota más alto o más bajo que lo que su nombre indica. Para hacer esto se utilizan los símbolos «agudo» # y «grave» b que indican la subida de un tono en un semitono y la bajada de un tono en un semitono. Cuando se vea un símbolo agudo o grave únicamente hay que calcular el valor del tono para la nota, como es normal, y sumarle o restarle 1 para hacerla más aguda o más

grave. Hay dos fomas de indicar las agudas y las graves: escribiendo los símbolos al lado de las notas determinadas o en la «clave» al comienzo del pentagrama. Si los símbolos se encuentran al comienzo de la música, el efecto de hacerlas más agudas o más graves afecta a *todas* las notas de la pieza musical. Y no sólo afecta a las notas que están en la misma posición en las líneas de los bordes, como el símbolo de llana o grave; también afecta a todas las notas con el mismo nombre. Es decir, el hacer más llana o más grave una *nota* escrita al comienzo de la partitura musical, afecta a la nota y a sus *octavas* durante toda la pieza musical. Pero si se escribe un símbolo de llana o grave al lado de una nota determinada de la composición musical, el efecto de hacerlas más agudas o más graves afecta únicamente a esa nota.

El segundo aspecto de la notación musical (la duración de cada nota) es más fácil de entender. Todas las notas duran un múltiplo de una unidad básica de tiempo. Si suponemos que una nota «normal» (una negra) dura una unidad de tiempo, cada cola oblicua dibujada en la nota divide por dos el tiempo que dura la nota (ver figura 3). También hay notas que duran más que la negra, comenzando con la blanca y la redonda, que duran dos y cuatro unidades de tiempo respectivamente. Si se observa una partitura musical auténtica se verá que algunas notas están agrupadas mediante sus colas. Estos y otros signos de frases y acentos no tienen ninguna importancia real cuando se pasa la música al Spectrum, pues no puede variar

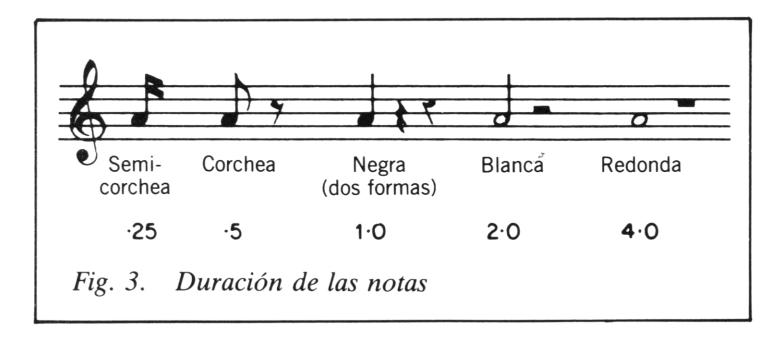



*Fig. 3. Duración de las notas*

el volumen de la nota que genera. La otra característica sobre la duración de las notas que es preciso conocer, aparte de la información de la figura 3, es que un punto que sigue a una nota aumenta su duración en la mitad de su duración natural. Por ejemplo, una negra seguida por un punto durará $1^{1}/_{2}$ unidades de tiempo. Las pausas musicales se indican mediante los símbolos mostrados en la figura 3.

## El Spectrum interpreta un tono

Después de toda esta teoría sobre las notaciones musicales, conviene ver un ejemplo. En la figura 4 se pueden ver las primeras líneas de la familiar melodía tradicional americana, «Dixie».

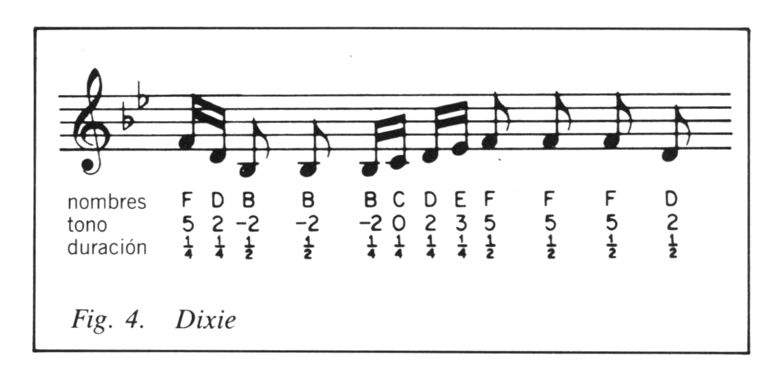



*Fig. 4. Dixie*

Lo primero que hay que hacer es escribir debajo de cada una de las notas su nombre, luego hay que convertir los nombres de las notas en valores de tonos, sin olvidarnos de tener en cuenta las llanas del principio de la música, y por último, escribir las duraciones en forma de fracciones de la unidad de tiempo.

Después de toda esta codificación, sólo falta escribir el programa. Hay muchas maneras de escribir un programa, pero posiblemente una de las más versátiles es utilizar una

instrucción DATA. En la instrucción DATA se escribe cada pareja de números (tono y duración), y el final de la música se indica mediante una pareja de valores como 99,99 que carecen de sentido como valores de tono o de duración. El programa completo incluyendo el resto de «Dixie» es:

```
10 DATA 5,.25,2,.25,-2,.5,-2,.5,-2,.25
20 DATA 0,.25,2,.25,3,.25,5,.5,5,.5,5,.5
30 DATA 2,.5,7,.5,7,.5,7,.5,5,.5,7,.5
40 DATA 7,.5,7,.25,9,.5,10,.25,12,.25
50 DATA 14,1.5,10,.25,5,.25,10,1.5,5,.25
60 DATA 14,.25,5,1.5,12,.25,14,.25,10,1.5
70 DATA 5,.25,5,.25,10,.5,14,.5,12,.5
80 DATA 10,.5,7,.5,10,1,10,.5,12,1.5,7,.5
90 DATA 12,1.5,5.,5,10,.5,14,.5,12,.5,10,.5
100 DATA 7,.5,9,.5,10,.75,7,.25,7,.25
110 DATA 5,.5,10,.5,2,.5,2,.5,0,1,2,.5
120 DATA -2,.75 2,.5,0,.75,7,.5,5,.5,2,.5
130 DATA 10,.75,14,.25,12,.5,10,1,99,99
140 LET d=.5
150 READ p,t
160 IF p=99 THEN STOP
170 BEEP d*t.p
180 GOTO 150
```

La técnica para programar otras melodías es la misma. Si es necesario incluir un intervalo, se puede codificar como un valor de tono absurdo seguido por el tiempo de duración de la pausa.

### Interpretar música

La otra cara de la moneda de la programación del Spectrum para interpretar música es escribir un programa que convierta al ordenador en un instrumento musical. En tal caso, se podría utilizar el Spectrum para interpretar música ya existente o incluso para componer música. La idea básica es fácil

de ejecutar. Se puede utilizar la función INKEY$ para leer el teclado y ver si se está pulsando una tecla; en caso afirmativo, lo único que hay que hacer es mostrar el tono al que se quiere que corresponda la tecla. Y ahí radica la dificultad: asignar los varios tonos y semitonos a un teclado con un teclado de máquina de escribir es algo muy difícil de hacer si se quiere tener un «instrumento» fácil de tocar. De todas formas, a título de ejemplo, ejecutar el siguiente programa:

```
10 DATA 0,2,4,5,7,9,11,12
20 DIN n(8)
30 FOR i=1 TO 8
40 READ n(i)
50 NEXT i
60 LET a$=INKEY$
70 IF a$="" THEN GOTO 60
80 BEEP .2,n(VAL(a$(1)))
90 GOTO 60
```

que nos permitirá interpretar la escala C utilizando las teclas de números de la hilera superior. Las líneas 10 a 50 seleccionan una matriz «n» que contiene los valores de las 8 notas. Las líneas 60 y 70, examinan el teclado hasta que se pulse una tecla; cuando se pulse una tecla, la línea 80 generará una nota de la duración determinada y con el tono correcto. Esto se consigue al utilizar la función VAL, que transforma la matriz a$ en un número que sirve para «indexar» la matriz «n».

Se debe ser capaz de ampliar este programa e incluir otras teclas, agudas y graves, sin demasiada dificultad, pero el que sea todavía más ambicioso tendrá que abandonar el BASIC y utilizar el código máquina para acelerar las cosas.

## Música automática

También puede ser atractiva la música automática. Cualquier cosa que sea muy complicada queda fuera del alcance

del Spectrum y de este libro, pero se pueden sacar a la luz algunas innovaciones interesantes.

Y también puede ser atractiva la música totalmente aleatoria:

```
10 BEEP RND,INT(20-40*RND)
20 GOTO 10
```

Esta música puede resultar interesante, pero no por mucho tiempo. Para que una secuencia de sonidos se convierta en música hay que imponer algún tipo de orden. Se puede conseguir un ligero perfeccionamiento haciendo que el valor del tono cambie solo una unidad por cada nota que se interpreta.

```
10 LET p=INT(40-80*RND)
20 BEEP .1,p
30 LET p=p+SGN(.5-RND)
40 GOTO 20
```

La línea clave en este programa es la línea 30, que suma o resta al azar 1 a p. El resultado es ligeramente más satisfactorio que el del programa anterior, pero incluso éste se vuelve tedioso al cabo de un rato. Aparte de lo señalado anteriormente, no hay mucho más por hacer en el campo de la música compuesta por ordenador: continúa en estado de investigación.

## Aumentar el sonido

Puede que (durante esta fase) algunas personas estén cansadas de intentar oír el ruido generado por el diminuto altavoz del Spectrum. Los que se encuentran en esta situación, se alegrarán de saber que hay una forma de escuchar la música del Spectrum a un volumen mucho más alto sin ningún equipo especial. El sonido que genera BEEP está presente en los enchufes MIC y EAR de la parte posterior del Spectrum. Lo

único que hay que hacer es preparar el registrador de cintas, que se utiliza para grabar los programas SAVE y LOAD, y cuando el Spectrum genere una pieza musical o cualquier otro sonido quedará registrado en cinta; una vez registrado se puede rebobinar la cinta y poner el volumen que se quiera. Pero si en realidad uno está interesado en oír el Spectrum a todo volumen, se puede conectar un amplificador y un altavoz externo al enchufe EAR.

Otra aplicación técnica del Spectrum es la generación de sonido. En realidad, la señal procedente del enchufe EAR es una onda cuadrada; una expresión bastante complicada con una frecuencia determinada por:

$$f=2^{p/12} \times fc$$

p es el valor del tono y fc es la frecuencia de C media, que es 261,6 ciclos por segundo, o Hercios, como es conocida actualmente. Aunque es una expresión complicada, el Spectrum la puede calcular muy fácilmente, ¡para qué sirven los ordenadores si no! El programa siguiente calculará la frecuencia producida por cualquier valor de tono:

```
10 INPUT "valor del tono=";p
20 LET f=261.6*2^(p/12)
30 PRINT "frecuencia=";f;" Hz"
40 GOTO 10
```

y el siguiente programa dará el valor del tono para cualquier frecuencia.

```
10 INPUT "frecuencia en Hz=";f
20 LET p=12*LN(f/261.6)/LN(2)
30 PRINT "valor del tono=";p
40 GOTO 10
```

No hay ninguna razón especial para que el valor del tono de una orden BEEP tenga que ser un número entero. Por

ejemplo, un valor del tono de 0,25 genera un cuarto de tono por encima de C media. Se puede generar cualquier tono utilizando valores de tono fraccionarios.

**Sonido y movimiento**

Como se señaló anteriormente, no es posible generar sonido y hacer otra cosa al mismo tiempo. Concretamente, es imposible acompañar con sonido los gráficos dinámicos. El Spectrum puede estar o moviendo algo o haciendo un ruido, pero no las dos cosas. La única solución es mover algo durante un momento, hacer algún ruido, volver a moverlo y así continuamente. No es una solución completamente satisfactoria pero es la mejor que se puede obtener en BASIC. Por ejemplo, si se quiere que el lanzamiento de un misil esté acompañado por sonido, lo único que podemos hacer es lo siguiente:

```
10 LET y=21
20 PRINT AT y,15;"^";
30 BEEP .001,50-y*2
40 LET y=y-1
50 IF y=0 THEN STOP
60 PRINT AT y+1,15;" "
70 GOTO 20
```

Las líneas 20 y 60 animan un misil (el símbolo de una flecha hacia arriba) desde la parte inferior a la superior de la pantalla, y la línea 30 genera un BEEP cada vez que se mueve el misil. Como el tono del sonido aumenta a medida que el misil se mueve hacia la parte superior de la pantalla, da la sensación de que el sonido viene de la figura en movimiento.

Se puede utilizar la misma técnica para producir sonido y movimiento con un razonable índice de éxito, siempre que el tiempo entre cada movimiento sea corto; en caso contrario, lo que se oye es una serie de sonidos desconectados que pa-

recen no tener nada que ver con el movimiento. Una excepción a esta regla es el ocasional ruido de un pitido que se utiliza para acompañar el rebote de una pelota. La regla, en este caso, es retener el sonido lo menos posible. Durante el tiempo que dura el pitido la pelota no se puede mover, y una pelota que se para cada vez que rebota resulta algo rara. Si se quiere ver la diferencia probar a borrar la orden BEEP del programa de squash del Capítulo 6.

Muchas veces es tentador crear un sonido en un programa a la mínima oportunidad. El resultado puede ser, desde el punto de vista del usuario, un programa muy ruidoso e irritante. Hay que utilizar el sonido sólo cuando proporciona algo interesante al programa. Utilizarlo para acompañar acciones como la pulsación de una tecla, el rebote de una pelota, el acierto en un blanco, o para atraer la atención del usuario hacia un mensaje de error de la pantalla. La utilización excesiva del sonido va en detrimento de su efectividad.

**Efectos sonoros**

No se puede hacer demasiado con el Spectrum, desde el BASIC, para generar efectos de sonido convincentes. No obstante, hay una forma de controlar el movimiento del altavoz directamente desde el BASIC. El equipo externo, como la impresora ZX, es controlado por el Spectrum utilizando dos órdenes adicionales, IN y OUT. Para distinguir entre las diferentes piezas del equipo externo, cada una tiene un número: su dirección. La orden

```
IN dirección
```

lee la información del dispositivo externo correspondiente a la «dirección». Y

```
OUT dirección, valor
```

envía el número que proporciona «valor» al dispositivo externo correspondiente a «dirección». En BASIC también están presentes las órdenes IN y OUT y se pueden utilizar para controlar directamente los dispositivos externos siempre que se sepa la dirección del dispositivo.

El altavoz del Spectrum es un dispositivo externo que tiene como dirección 254, la cual se puede utilizar en la orden OUT para que el altavoz genere un chasquido. La única complicación adicional es que el color del margen que rodea la pantalla, también es controlado como un dispositivo externo; y con la misma dirección. Si se quiere ver el efecto de esta confusión probar el siguiente programa:

```
10 FOR i=0 TO 255
20 OUT 254,i
30 NEXT i
```

Algunas veces cambiará de color el margen de la pantalla y otras veces se oirá un chasquido del altavoz. La solución de este problema está en el hecho de que el valor que se envía al dispositivo externo controla el color o el altavoz según la gama en la que se encuentra. Por ejemplo, los números que van del 0 al 15 afectan únicamente al color del margen. Técnicamente, los tres primeros bits del valor especifican el color del margen y el cuarto bit controla el altravoz. Lo único que nos falta por saber es que la posición de memoria 23624 contiene ocho veces el color del margen actual. La utilización de toda esta información para provocar un chasquido en el altavoz, nos da como resultado el siguiente programa:

```
10 LET a=PEEK 23624/8
20 OUT 254,a-16
30 OUT 254,a
```

La primera línea de este programa lee el color actual del margen (en el Capítulo 7 se describe con más detalle la orden PEEK). La primera OUT, línea 20, cambia el estado del al-

tavoz y la línea 30 lo restaura a su estado original. Cada vez que se ejecuta el programa genera un solo chasquido. Si se compone un bucle añadiendo 40 GOTO 20 al final del programa, el chasquido se convierte en un sonido bajo y áspero. La calidad de este sonido puede ser modificada añadiendo órdenes adicionales entre las líneas 20 y 30, para crear un retraso. La solución más sencilla es añadir un número variable de instrucciones REM. Con un poco de práctica se estará en condiciones de generar el sonido de una ametralladora.

Otras técnicas para crear efectos sonoros se basan en la utilización de chasquidos junto con otros sonidos. Por ejemplo, un chasquido seguido por un sonido de tono alto suena como si se golpeara algo. Los sonidos de tono muy bajo pueden servir para simular explosiones, etc. Es sorprendente lo que se puede conseguir utilizando únicamente un chasquido y un BEEP. El que sea ambicioso tendrá que pasar al campo del código máquina, pero incluso ahí el método continúa siendo básicamente el mismo. La única diferencia es el número de chasquidos que se pueden generar por segundo: se pueden generar muchos más en código máquina que en BASIC.

# 6. GRAFICOS EN MOVIMIENTO

Uno de los campos más agradecidos de la informática son los gráficos dinámicos o en movimiento. No es ni mucho menos obvia la forma de pasar del trazado de un solo punto en cualquier sitio de la pantalla a la creación de una representación visual en movimiento. En realidad la transición es bastante fácil.

## De la intermitencia al movimiento

Si se traza un solo punto y luego se vuelve a «destrazar», se verá un pequeño punto que brilla intermitentemente. Hacer la prueba en el siguiente programa:

```
10 PLOT 100,50
20 PLOT INVERSE 1,100,50
30 GOTO 10
```

(Habrá que mirar detenidamente para ver el pequeño punto que brilla intermitentemente en la mitad de la pantalla). Se

puede conseguir el mismo efecto imprimiendo un bloque gráfico (ˆ8) y luego, en el mismo sitio, un espacio vacío de gráficos (8). Porbar con:

```
10 PRINT AT 10,10;"[ˆ8]";
20 PRINT AT 10, 10;"[8]";
30 GOTO 10
```

El cuadrado intermitente generado por la orden PRINT es ocho veces mayor que el generado por PLOT, y su intermitencia es más rapida, ya que el Spectrum tiene que hacer mucho menos trabajo para ejecutar una PRINT que para ejecutar una orden PLOT.

En ambos casos, para alterar la velocidad de intermitencia hay que añadir bucles de retardo. Y decimos BUCLES de retardo porque si se intenta colocar un solo bucle FOR (por ejemplo, en la línea 15), aumentará el tiempo que el punto está «encendido» pero no aumentará el tiempo que está «apagado». Para prolongar el tiempo que el punto está «apagado» también se necesita un bucle FOR en la línea 25. Por ejemplo, hacer la prueba con:

```
10 INPUT tiempo de presencia
20 INPUT tiempo de ausencia
30 PRINT AT 10,10;"[ˆ8]";
40 FOR i=1 TO tiempo de presencia
50 NEXT i
60 PRINT AT 10,10;"[8]";
70 FOR i=1 TO tiempo de ausencia
80 NEXT i
90 GOTO 30
```

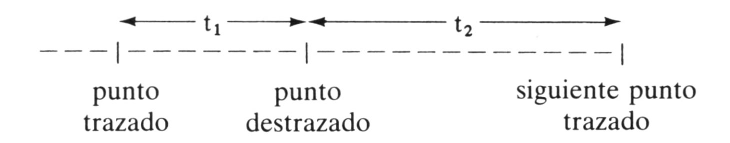
Ahora sabemos todo lo que hay que saber sobre la manera de hacer brillar cosas intermitentemente, pero ¿y el movimieno? En realidad, el paso de la intermitencia al movimiento es fácil. Si se traza un punto y se «destraza» y luego se traza el siguiente punto, dará la sensación visual de que el «punto» se

ha «movido». Si se continúa repitiendo este proceso se puede crear la sensación de que el punto parece moverse continuamente. Por ejemplo, supongamos que queremos mover un punto desde un lado de la pantalla a otro en línea recta. En el capítulo anterior se mostró la forma de crear una línea recta, pero en un principio intentemos algo más fácil. Si el punto se mueve horizontalmente, simplemente hay que aumentar la abscisa cada vez. Probar el siguiente programa:

```
10 LET y=50
20 FOR x=0 TO 255
30 PLOT x,y
40 PLOT INVERSE 1,x,y
50 NEXT x
```

Funciona exactamente de la forma descrita. La línea 30 traza un punto, la línea 40 lo «destraza», la línea 30 traza el siguiente punto contiguo, y así sucesivamente.

Cuando se utiliza este método hay que comprobar que todo sucede justo en el momento adecuado para dar la impresión de movimiento a la velocidad deseada. En este ejemplo sencillo hay dos tiempos que importan: el tiempo que transcurre entre trazar un punto y «destrazarlo», y el tiempo que transcurre entre «destrazar» el punto anterior y trazar uno nuevo. Un diagrama puede ayudar a clarificar lo expuesto:




El tiempo $t_1$ es el tiempo que está presente en la pantalla *algún* punto y $t_2$ es el tiempo en que no hay *ningún* punto visible en la pantalla. El tiempo total, $t_1 + t_2$, es el tiempo que tarda el punto en moverse de una posición a otra; rige la rapidez con la que parece moverse.

La mayoría de los libros de gráficos en movimiento no especifican qué valores deben tener $t_1$ y $t_2$ para generar una imagen fluida. La respuesta no es fácil, y es una práctica normal tantear los valores del programa de $t_1$ y $t_2$ hasta conseguir la mejor imagen posible. Es fácil ver qué valores deben tener $t_1$ y $t_2$ en teoría. Si estuviéramos observando un punto en movimiento detrás de una rejilla de agujeros, el tiempo que el punto fuera visible correspondería a $t_1$ y el tiempo que el punto estuviera oculto correspondería a $t_2$. Si los agujeros de la rejilla estuvieran más juntos y a la misma distancia, todavía podríamos ver el punto en «movimiento», porque el cerebro humano tiende a interpretar una secuencia de imágenes como movimiento. Lo que en realidad hacemos con el punto en movimiento intermitente en la pantalla del Spectrum es copiar el principio de un objeto escondido detrás de una rejilla de agujeros y confiar en la realidad de que el cerebro se deja engañar y ve movimiento. La calidad del movimiento aparente en la pantalla se puede relacionar con la exactitud con que se copia lo que se ve a través de la rejilla. Si los agujeros están muy juntos, será mayor el tiempo que estará visible el punto y menor el tiempo que estará oculto. Dicho de otra forma, $t_1$ será mucho mayor que $t_2$. Esta es la condición para crear movimientos fluidos en la pantalla del Spectrum. Desafortunadamente, no es nada fácil de conseguir. El Spectrum tarda lo mismo en trazar que en destrazar, por lo que $t_2$ tiende a ser igual que $t_1$. Y además el método que estamos utilizando empeora las cosas. Se traza un punto, se destraza, y luego se hacen algunos cálculos para volver a trazarlo, lo que significa que $t_2$ es mucho mayor que $t_1$. El resultado de este desequilibrio de los tiempos de «encendido» y «apagado» es que el punto en movimiento tiende a «parpadear» o brillar intermitentemente cuando se mueve. Esto se podría perfeccionar trazando el punto, haciendo los cálculos, destrazando el punto anterior y volviendo a trazar el punto nuevo. El problema está en que esta solución exige almacenar las posiciones anterior y siguiente del punto trazado, lo que por de pronto dificulta un poco la comprensión de los programas. Para decidir si merece la pena este perfeccionamiento, hacer la prueba con:

```
10 LET y=50
20 FOR x=0 TO 255
30 PLOT INVERSE 1,x,y
40 PLOT x+1,y
50 NEXT x
```

Un método más sencillo para conseguir más fluidez en el movimiento es aumentar $t_1$ añadiendo un elemento de pérdida de tiempo, como 35 LET y=y,entre las instrucciones de trazar y destrazar. Es el mejor método para el Spectrum, lo que no obsta para que alguien esté interesado en probar con otros métodos.

Hay otra cosa que se puede aprender meditando sobre la visualización de un punto en movimiento a través de una rejilla de agujeros. Si el punto se mueve a una velocidad S y resulta invisible durante un tiempo $t_2$, la distancia entre los agujeros es $S*t_2$. Esto sugiere que, como nuestro problema con el Spectrum es que $t_2$ es demasiado grande para una imagen fluida, se puede perfeccionar aumentando la distancia entre los puntos de la imagen. Esto se puede conseguir simplemente trazando un punto, «destrazándolo» luego, y *en vez* de trazar un punto contiguo, trazándolo más lejos. Probar el siguiente programa:

```
10 LET y=10
20 FOR i=1 TO 10
30 FOR x=1 TO 31 STEP i
40 PRINT AT y,x;"[ˆ8]";
50 PAUSE 5
60 PRINT AT y,x;"[8]";
70 NEXT x
80 NEXT i
```

Cada vez que se ejecuta el bucle 30-70 se mueve un punto por la pantalla de izquierda a derecha. La primera vez la distancia entre los puntos trazados es 1, la próxima es 2, y así sucesivamente hasta que la distancia es 10. Lo que resulta intere-

sante de este ejemplo es que, incluso cuando el punto salta seis o siete puntos de un solo golpe, continúa ofreciendo la ilusión de movimiento; el movimiento de máxima fluidez se consigue con un paso aproximado de 2 o 3.

Aunque el paso de este movimiento más largo resulta interesante, no siempre es eficaz, porque muchos programas de gráficos dinámicos necesitan mover el punto únicamente un paso cada vez.

## El rebote de pelotas y la velocidad

Ahora que se ha mostrado la forma de hacer que un punto se mueva por la pantalla, pensemos la forma de utilizarlo con fines más interesantes y excitantes. En primer lugar, se podría trazar y volver a trazar el perímetro de un círculo o de un cuadrado para conseguir que un punto se mueva no sólo en línea recta. No obstante, analicemos una aplicación más realista. En los juegos dinámicos puede resultar más práctico simular el movimiento de una pelota, y la mejor forma de hacerlo es definiendo las dos velocidades a las que se mueve la pelota. En cada paso de movimiento, el punto trazado (o pelota) se puede mover un número de lugares horizontalmente y un número de lugares verticalmente. Cada paso tarda la misma cantidad de tiempo, luego podemos llamar a la distancia que se mueve horizontalmente «velocidad horizontal», y a la distancia que se mueve verticalmente «velocidad vertical». De esta forma en cada paso de movimiento la velocidad horizontal se suma a la abscisa y la velocidad vertical se suma a la ordenada. Probar el siguiente programa:

```
10 LET v=1
20 LET h=1
30 LET x=0
40 LET y=0
50 PRINT AT y,x;"[^8]"
60 PRINT AT y,x;"[8]"
```

```
70 LET x=x+h
80 LET y=y+v
90 GOTO 50
```

Este programa mueve una pelota desde la parte superior izquierda de la pantalla a la parte inferior derecha y después la saca de la pantalla. Como la pelota sale disparada fuera de la pantalla el programa finaliza con un error. Lo lógico es dejar que la pelota rebote en los bordes de la pantalla, pero ¿como? La respuesta es sorprendentemente fácil, porque hemos decidido aplicar la idea de la velocidad vertical y horizontal. Si la pelota encuentra una pared vertical, por ejemplo el borde derecho de la pantalla, no puede continuar moviéndose en la misma dirección horizontal. En realidad, sólo un cambio completo de la velocidad horizontal evitará que traspase la pared. La velocidad vertical no resulta afectada cuando encuentra una pared vertical. ¿Por qué debería resultar afectada? La regla es: cuando la pelota encuentra una pared vertical invierte la velocidad horizontal, y cuando la pelota encuentra una pared horizontal invierte la velocidad vertical. Aplicando estas dos reglas se obtiene el siguiente programa:

```
10 LET v=1
20 LET h=1
30 LET x=0
40 LET y=0
50 PRINT AT y,x;"[^8]"
60 PRINT AT y,x;"[8]"
70 LET x=x+h
80 LET y=y+v
90 IF x=0 OR x=31 THEN LET h=-h
100 IF y=0 OR y=21 THEN LET v=-v
110 GOTO 50
```

Este es un programa extraordinariamene sencillo para los efectos que consigue. Las líneas 90 y 100 supervisan la presencia de una pared horizontal o vertical, y si encuentran alguna, invierten la velocidad (invertir la velocidad es lo mismo

que poner un signo menos delante de la velocidad anterior). Debido a las diferentes formas de numerar las posiciones de la pantalla, si se están utilizando instrucciones PRINT AT las velocidades positivas nos llevan desde la *parte superior* a la parte *inferior* de la pantalla, pero si se están utilizando instrucciones PLOT nos llevarán desde la *parte inferior* a la *parte superior*.

Aunque este programa resulta ya impresionante, todavía se puede mejorar más. En primer lugar, la pelota en movimiento está en estado intermitente más de lo necesario. Se puede aumentar el tiempo entre la aparición de la pelota y su desaparición de la pantalla pasando la línea 50 a la 85 (sin olvidar cambiar la línea 110 por GOTO 60). Si se intenta con este programa, se obtendrá un movimiento mucho más fluido. La siguiente mejora es inclùir una orden BEEP cada vez que la pelota rebota. Para hacerlo, cambiar las líneas 90 y 100 por éstas:

```
90 IF x=0 OR x=31 THEN LET h=h-1:BEEP .01,10
100 IF y=0 OR y=21 THEN LET v=v-1:BEEP .01,10
```

Este cambio sencillo resulta muy interesante, ya que parece hacer más positivo el rebote. Sin embargo, también nos alerta de otro problema: la pelota parece detenerse cada vez que rebota. Esto se verá muy claramente si se quiere cambiar la duración de la BEEP de 0,01 a 0,1. La razón es bien sencilla: hay que hacer muchos más cálculos si la bola rebota que en caso contrario. Para ajustar más las cosas hay que incluir un retraso en el movimiento normal de la pelota que sea igual al tiempo que tarda la pelota en rebotar. Dicho de otra forma, hay que hacer que cada paso de movimiento tarde la misma cantidad de tiempo, tanto si rebota como si no. El programa final es:

```
10 LET v=1
20 LET h=1
30 LET x=0
```

```
40 LET y=0
60 PRINT AT y,x;"[8]"
70 LET x=x+h
80 LET y=y+v
85 PRINT AT y,x;"[^8]"
90 IF x=0 OR x=31 THEN LET h=-h:BEEP .01,10:GOTO 60
100 IF v=0 OR y=21 THEN LET v=-v:BEEP .01,10:GOTO 60
105 PAUSE 3
110 GOTO 60
```

(Obsérvese que las líneas de este programa no están numeradas en saltos uniformes de 10, con objeto de poder compararlo con el programa anterior). La PAUSE 3 de la línea 105 entra en acción únicamente si la bola no rebota. La duración de la PAUSE se regula mediante un método de tanteo para hacer que la pelota se mueva lo más fluidamente posible. Sin embargo, hay que tener en cuenta que hace que la ejecución del programa completo resulte mucho más lenta: la perfección exige un precio.

Ahora que se sabe mover y hacer rebotar una pelota, puede parecer lógico intentar escribir un juego del tipo bate-pelota. Después de todo, la acción de golpear la pelota con un bate se ajusta a las reglas de la inversión de la velocidad igual que la pelota que golpea contra una pared. ¡Por supuesto, este será nuestro próximo paso!

**Squash**

A título de ejemplo de un programa de «rebote de pelota», examinar el siguiente programa de squash:

```
10 LET pelota=0
20 GOSUB 800
30 INK 0:PAPER 6:CLS
40 LET a=10:LET b=10
50 LET v=1:LET w=1
```

```
60  LET x=10:LET y=21
70  LET pelota=pelota+1
80  GOSUB 500
90  PRINT pelota
100  GOSUB 200
110  GOSUB 700
120  GOSUB 300
130  IF b<>21 THEN GOTO 100
140  BEEP .5,-10
150  GOTO 30

200  LET a$=INKEY$
210  IF a$="5" AND x>0 THEN LET x=x-1
220  IF a$="8" AND x<27 THEN LET x=x+1
230  RETURN

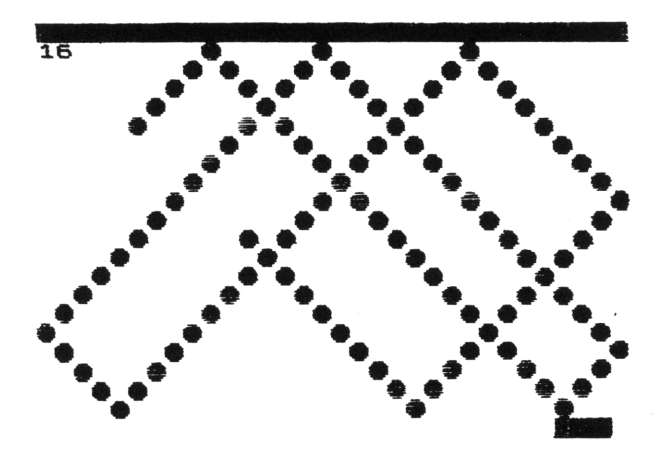
300  PRINT AT b,a;"[8]";
310  LET a=a+v
320  LET b=b+w
330  IF a=31 OR a=0 THEN LET v=-v:BEEP .01,10
340  IF b=1 THEN LET w=-w:BEEP .01,10
350  IF b+1=y THEN GOSUB 400
360  PRINT AT b,a;"[a]";
370  RETURN

400  LET r=a-x
410  IF r<1 OR r>3 THEN RETURN
420  LET w=-w
430  BEEP .01,0
440  RETURN

500  FOR i=0 TO 31
510  PRINT AT 0,i;"[^8]";
520  NEXT i
530  RETURN

700  PRINT AT y,x;"[8][^8][^8][^8][8]";
710  RETURN
```

```
800 POKE USR "a"+0,BIN 00111100
810 POKE USR "a"+1,BIN 01111110
820 POKE USR "a"+2,BIN 11111111
830 POKE USR "a"+3,BIN 11111111
840 POKE USR "a"+4,BIN 11111111
850 POKE USR "a"+5,BIN 11111111
860 POKE USR "a"+6,BIN 01111110
870 POKE USR "a"+7,BIN 00111100
880 RETURN
```




Es una aplicación sencilla de los métodos de gráficos en movimiento que han sido explicados anteriormente. El programa comienza definiendo una forma de pelota en la subrutina 800 y luego imprime la pared de la pista mediante la subrutina 500. La subrutina 300 sitúa la pelota en la

posición a,b y sigue la trayectoria de la velocidad horizontal v y de la velocidad vertical w de la pelota. Si la pelota golpea una «pared», (intenta salirse de la pantalla), la velocidad correcta es modificada para hacerla rebotar. La subrutina 700 imprime el bate en la posición x,y. Observar que no es necesario borrar el bate anterior, debido a los dos espacios en blanco en cada punta del bate. Las dos características nuevas

son: la forma en que se mueve el bate y la forma en que la pelota rebota al entrar en contacto con el bate. Para mover el bate lo único que hay que hacer es cambiar la coordenada horizontal del bate de acuerdo con la tecla que se pulse; si se pulsa la tecla de la flecha izquierda, se resta uno de x y se desplaza el bate un lugar a la izquierda, y si se pulsa la tecla de la flecha derecha se suma uno a x y se desplaza el bate un lugar a la derecha. Todo esto se hace mediante la subrutina 200. Antes de actualizar la posición del bate se realiza una supervisión para comprobar que el bate está todavía en la pantalla. La subrutina 400 tiene a su cargo el rebote de la pelota al golpear el bate. Si la pelota está en la misma posición vertical que el bate, la línea 350 solicita a la subrutina 400 que compruebe si está en la posición horizontal correcta para el rebote, para invertir su velocidad vertical. Si la pelota sale fuera de la pantalla por la parte inferior (se detecta mediante la línea 130), el control pasa a la línea 100 y se inicia un nuevo juego. No hay que olvidar que el juego es ejecutado por las líneas 100 a 130 repetidas constantemente. La línea 100 llama a la rutina del movimiento del bate, la línea 110 es la que realmente mueve el bate y la línea 120 llama a las rutinas que vigilan la pelota y sus rebotes.

## El vuelo libre y la gravedad

El apartado anterior analiza el movimiento de una pelota en el interior de un marco y la forma de hacer que rebote. Hay otra forma de mover una pelota: se puede lanzar al aire. Intentemos buscar una forma de hacer que una pelota se mueva bajo la influencia de la gavedad.

En el espacio sideral, donde no hay gravedad, una pelota puesta en movimiento en una dirección determinada y a una velocidad determinada, continuará moviéndose en la misma dirección y a la misma velocidad eternamente (a menos que choque con algún objeto, lo que produciría un rebote en la dirección contraria pero a la misma velocidad, como sucede

con la pelota en el apartado anterior). En este sentido, la forma que, de momento, se conoce de mover una pelota corresponde al movimiento libre de gravedad. Escribamos un programa que simule el lanzamiento de una pelota en un espacio sin gravedad.

```
10 LET v=0
20 LET h=1
30 LET x=0
40 LET y=0
50 PRINT AT y,x;“[ˆ8]”
60 PRINT AT y,x;“[8]”
70 LET x=x+h
90 LET y=y+v
100 GOTO 50
```

Si se miran las líneas 10 y 20 se podrá ver que la pelota es lanzada horizontalmente hacia abajo desde la parte superior de la pantalla. Es como si dejáramos caer una pelota desde un acantilado.

Si se introduce gravedad, la diferencia consiste en que la velocidad vertical cambia. Por ejemplo, si se suelta una pelota, cae, y su velocidad vertical aumenta a medida que cae. Dicho de otra forma, cuando la pelota se mueve una unidad horizontalmente su velocidad vertical aumenta una cantidad determinada. El valor de la cantidad determinada depende de la fuerza de la gravedad, pero para nuestros propósitos lo podemos ajustar de forma que proporcione un resultado razonable. Para ver la caída de la pelota añadir la línea 80:

```
80 LET v=v+.1
```

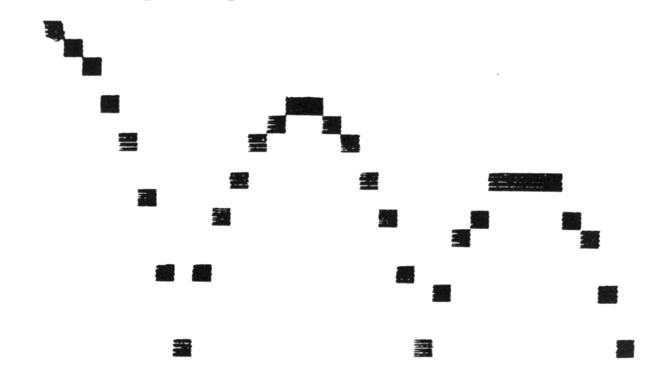
al programa de la «pelota libre». En su ejecución el programa imita a una pelota cayendo según una curva parabólica. El programa indica un error tan pronto como la pelota «cae» fuera de la parte inferior de la pantalla. Si se quiere perfeccionar el programa inténtese restando una pequeña cantidad a la velocidad horizontal para incluir la resistencia del viento.

Ahora podemos combinar lo que ya sabemos sobre el rebote de pelotas con lo que acabamos de descubrir sobre la gravedad. Si definimos una pared vertical en, por ejemplo, y=15, al alcanzar la pelota esa posición se puede aplicar la regla del «rebote» e invertir la velocidad vertical. El programa es:

```
10 LET v=0
20 LET h=1
30 LET x=0
40 LET y=0
50 PRINT AT y,x;"[ˆ8]"
60 PRINT AT y,x;"[8]"
70 LET x=x+h
80 LET v=v+.6
90 LET y=y+v
100 IF y>15 THEN LET v=-v
110 GOTO 50
```

Si se suprime la línea 60 el resultado se parece más o menos al que se muestra a continuación.

Ya se debe tener una buena idea sobre la forma de hacer lo que se quiera con una pelota. Utilizando la idea de la velocidad vertical y horizontal todo es sencillo. Si se quiere acelerar la pelota hay que sumar algo a la velocidad apropiada, o restar si lo que se quiere es reducir la velocidad.



**El alunizador**

Si se suman algunas adiciones al programa anterior de la caída de la pelota, se podrá crear un juego de aterrizajes en la luna razonable. El aterrizaje de un cohete en la luna funciona exactamente igual que la caída de la pelota, con la diferencia de que puede encender sus motores y reducir la velocidad vertical.

```
10 LET f=1200
20 GOSUB 300
30 LET b=0:LET v=0
40 LET h=RND
50 LET x=0
60 LET y=1
70 PRINT AT y,x;"[a]";
80 GOSUB 170
90 PAUSE 3:PRIN AT y,x;"[8]";
100 LET x=x+h
110 IF x>30 THEN GOSUB 250
120 LET y=y+v
130 IF y<19 THEN GOTO 70
140 IF v>.5 THEN PRINT AT 10,10;"**Explosión**"
150 IF v<=.5 THEN PRINT AT 10,3;"**El Spectrum ha alunizado**"
160 STOP

170 LET b$=INKEY$
180 IF b$="" THEN GOTO 200
190 LET b=VAL b$ *10
200 LET f=f-b
210 IF f<0 THEN LET b=0
220 LET v=v-b/1000+.05
230 PRINT AT 0,0;"H=";INT(11.6*(19-Y));"S=";INT(200*v);" F=";
    f;" BR=";b;" "
240 RETURN

250 LET x=0
260 CLS
270 GOTO 380
```

```
300 POKE USR "a"+0,BIN 00011000
310 POKE USR "a"+1,BIN 00111100
320 POKE USR "a"+2,BIN 01111110
330 POKE USR "a"+3,BIN 11111111
340 POKE USR "a"+4,BIN 00011000
350 POKE USR "a"+5,BIN 00111100
360 POKE USR "a"+6,BIN 00111100
370 POKE USR "a"+7,BIN 00000000
380 FOR i=0 TO 31
390 PRINT AT 21,i;"[3]";
400 NEXT i
410 RETURN
```

La línea 10 selecciona la cantidad de combustible que se tiene al comienzo del juego. Si se quiere hacer más fácil el juego, aumentar la cantidad de combustible a más de 1.200. El cohete comienza con una velocidad horizontal aleatoria, seleccionada en la línea 40, y cae por gravedad hasta que toca el suelo. A medida que cae, se puede quemar combustible para reducir su velocidad de descenso. Pulsando cualquier tecla de 0 a 9 se selecciona el ritmo al que se quema el combustible: el índice de combustión BR. El índice de combustión es diez veces el valor de la tecla pulsada. La línea 170 comprueba qué

```
H=120 S=180 F=1.200 BR=0
```

tecla es pulsada mediante la instrucción INKEY$. Continuar pulsando la tecla elegida porque unicamente afectará el programa una vez por cada movimiento del cohete. El combustible quemado es restado del combustible que queda; si se utiliza todo el combustible el cohete se precipita hacia la superficie. El objetivo del juego es aterrizar con una velocidad vertical de menos de 100 metros por segundo. Si se desplaza demasiado a la derecha se borra la pantalla y se vuelve a comenzar de nuevo desde el extremo izquierdo. ¡Feliz aterrizaje!

## El lanzamiento en una dirección determinada

A estas alturas, ya sabemos cómo se mueve una pelota bajo los efectos de la gravedad si se lanza horizontalmente desde un acantilado, pero en muchos juegos se necesita lanzar la pelota hacia arriba. Esto se puede conseguir invirtiendo la coordenada y de una orden PRINT AT, o simplemente poniendo en su lugar una instrucción PLOT. No hay que olvidar que PLOT 0,0 es la *parte inferior* izquierda y PRINT AT 0,0 es la *parte superior* izquierda. Ejecutar el siguiente programa.

```
10 LET x=0
20 LET y=0
30 LET h=2
40 LET v=4
50 PLOT x,y:PLOT x+1,y
60 PLOT INVERSE 1;x,y: PLOT INVERSE 1,x+1,y
70 LET x=x+h
80 LET v=v-.1
90 LET y=y+v
100 IF y<1 THEN STOP
110 GOTO 50
```

La velocidad inicial es h=2 y v=4. A cada paso, la velocidad vertical desciende 0,1. Por lo tanto, la pelota comienza moviéndose hacia arriba bastante rápida y luego reduce su ve-

locidad hasta que su movimiento es únicamente hacia adelante. Entonces se invierte la dirección vertical y la pelota comienza a caer hacia la parte inferior de la pantalla. Hay que tener también en cuenta que se han trazado dos puntos para facilitar su visión. El resultado es la forma de la conocida parábola del lanzamiento de un objeto.



Normalmente se lanza la pelota con un ángulo determinado y a una velocidad determinada. El lanzamiento de la pelota a una velocidad determinada controla su velocidad global; es decir, cuanto más fuerte se lance la pelota más rápida irá al principio. El ángulo con el que se lanza la pelota altera la distribución de su velocidad global entre las componentes vertical y horizontal. Por ejemplo, si se lanza la pelota recta a 90 grados, la pelota se mueve verticalmente pero no horizontalmente, y a medida que se disminuye el ángulo la pelota se mueve más horizontalmente y menos verticalmente. Si se analiza matemáticamente la situación se verá que: si se tira la pelota con una fuerza que da lugar a una velocidad total de v con un ángulo t, la velocidad horizontal está determinada por v*COS(t) y la velocidad vertical está determinada por v*SIN(t). Utilizando estos dos valores iniciales para las velocidades horizontal y vertical, se puede aplicar el mismo tipo de programa para mover la pelota bajo los efectos de la gravedad. Lo único que hay que tener en cuenta es que el Spectrum mide los ángulos en radianes (ángulo en radianes = ángulo en grados *PI/180).

### La bala de cañón

Ahora que sabemos lanzar algo con un ángulo determinado y con una fuerza determinada, se puede intentar escribir un programa de disparos. Hay un cañón preparado en el lado izquierdo de la pantalla y un blanco colocado aleatoriamente en el lado derecho. Hay que especificar dos números: el ángulo, de 0 a 90 grados, y la fuerza del disparo. E intentar acertar en el blanco. La fuerza del disparo es ilimitada pero los valores alrededor de 10 funcionan bien. Un problema adicional es que si se dispara con un ángulo que hace que la bola se salga de la pantalla por *cualquier sitio*, se cuenta como fallo, lo que significa que hay que disparar el cañón con un ángulo pequeño para tener la seguridad de no dar al borde superior de la pantalla. Esta restricción también evita que derribemos nuestras propias aeronaves. Los disparos de ángulo bajo son más difíciles, así que la limitación mejora en realidad el juego.

```
10 LEt b=30
20 GOSUB 300
30 INPUT "fuerza=";f
40 LET f=f/2
50 INPUT "ángulo=";t
60 LET h=f*COS(t*PI/180)
70 LET v=f*SIN(t*PI/180)
80 LEt x=0
90 LET y=b
100 PLOT x,y:PLOT x,y+1
110 LET v=v-.1
120 PLOT INVERSE 1,x,y:PLOT INVERSE 1,x,y+1
130 LET x=x+h
140 LET y=y+v
150 IF y<=b THEN GOTO 200
160 IF x>255 OR y>150 THEN GOTO 200
170 GOTO 100

200 IF x>=p AND x<=p+10 THEN GOTO 250
210 LET m=m+1
```

```
220 GOTO 260

250 BEEP .2,.10:LET s=s+1
260 PRINT AT 1,0;"aciertos=";s;"fallos=";m;" "
270 GOTO 30

300 LET p=RND*100+100
310 PLOT p,b
320 DRAW 10,0
330 LET m=0
340 LET s=0
350 RETURN
```

```
aciertos=1 fallos=3
```

La subrutina 300 traza el blanco en una posición aleatoria de 10 puntos de anchura e inicializa m y s, los contadores de fallos y de aciertos. Las líneas 30 a 50 dan entrada a f (la fuerza) y a t (el ángulo). La línea 40 gradúa la fuerza para ofrecer una razonable gama de velocidades. La sección central es simplemente el programa del lanzamiento de la pelota mostrado anteriormente, pero con instrucciones adicionales para comprobar si la pelota ha dado en el blanco.

Es posible escribir, además de estas líneas, programas mucho más complicados que permitan incluir elementos como la resistencia del aire y la dirección del viento ¡hay que experimentar!

## Resumen

Tenemos que estar ya suficientemente capacitados para amplicar y mejorar programas como el del aterrizaje lunar y el del squash y utilizarlos como base para nuestros propios juegos. Añadir sonido, marcadores y dificultades adicionales para los jugadores. Por ejemplo, se podría intentar escribir una versión del programa del squash para dos personas con dos bates.

# 7. PEEK Y POKE

Dos de las instrucciones más misteriosas de todo el BASIC son PEEK y POKE. La pregunta: «¿Para qué sirven las instrucciones PEEK y POKE?» es típica. La respuesta depende en gran medida del ordenador que se está utilizando. Este capítulo ofrece una breve explicación de la función de PEEK y POKE, y algunos ejemplos de la forma de utilizarlas en el Spectrum. No hay que esperar que lo que se aprenda aquí sobre la forma de utilizar PEEK y POKE se pueda emplear en otros ordenadores. La respuesta más segura es NO.

## La función de PEEK y POKE

De las dos instrucciones, la PEEK es la más fácil de entender y la más segura de utilizar. Es imposible ni siquiera arañar el ordenador con una PEEK incorrecta, pero desde luego es muy posible con la POKE. Aunque se haya tratado a la PEEK como instrucción, es más correcto llamarla «función» porque da un resultado. (Las funciones son elementos como SIN(x) que dan como solución algun número: el resultado). PEEK es un tipo especial de función que no «hace» ningún cálculo, simplemente «refleja» el contenido de una posición de memoria determinada y le pasa al sistema decimal. Por ejemplo:

```
10 LET a=PEEK 7688
```

seleccionará el contenido de la posición de memoria 7688. Hay muchas cosas que hay que saber sobre los ordenadores en general, y sobre el Spectrum en particular, para poder entender el sentido real del ejemplo anterior. En primer lugar, hay que saber que los ordenadores guardan y recuperan información en y de posiciones de memoria numeradas; cada posición tiene un solo número, conocido como su «dirección». En segundo lugar, hay que saber que la cantidad de información que se puede almacenar en cada posición es limitada; en el caso del Spectrum cada posición de memoria puede almacenar únicamente un carácter. Como es bien sabido, un ordenador únicamente puede almacenar en su memoria ceros o unos. ¿Cuál es entonces la forma de almacenar un carácter en una posición de memoria? La respuesta es que un grupo de bits (ceros o unos) puede ser interpretado como un número. Por ejemplo 0101 es cinco. No es importante de momento conocer la forma de convertir un grupo de bits en un número decimal, es suficiente con saber que se puede hacer. Un grupo de 8 bits es un byte: puede representar números desde 0 (00000000) hasta 255 (11111111), de forma que cualquier posición de memoria puede contener un número comprendido en esa gama. Después de toda esta explicación se debería tener una idea clara sobre el almacenamiento de caracteres en memoria como grupos de 8 bits. Si se mira en la contraportada del manual del Spectrum, se verá un listado del conjunto de caracteres del Spectrum. La primera columna tiene el título «código» y contiene números desde el 0 hasta el 255. Esto significa que se puede tratar al contenido de una posición de memoria como un número, por ejemplo 76, o como el carácter correspondiente (mediante CHR$(76)), que es la L. Lo más importante a tener en cuenta sin embargo, es que cualquier posición de memoria del Spectrum puede contener un número comprendido entre 0 y 255; si se mira el contenido de diversas posiciones de memoria se verán muchos números diferentes pero ninguno de ellos menor que 0 ni mayor que 255.

La otra cosa que hay que saber sobre la forma de empleo

de la PEEK es la gama de direcciones que pueden acompañarla. El Spectrum numera sus posiciones de memoria comenzando en el 0 y continuando hasta un máximo de 65535. No todas esas posiciones de memoria corresponden a algo en el Spectrum; como se verá más adelante, muchas de ellas están sin usar. Un último hecho que es importante conocer: hay dos tipos de memorias, RAM y ROM. La RAM, memoria de acceso aleatorio, puede ser utilizada para almacenar y presentar información. La ROM, memoria de sólo lectura, únicamente puede ser utilizada para presentar información. El Spectrum de 16K tiene cerca de 16000 posiciones de memoria ROM. Esta vasta cantidad de información integrada en cada Spectrum sirve para muchas cosas diferentes, pero una de sus principales aplicaciones es definir las reglas del lenguaje BASIC. La parte ROM de la memoria comienza en el 0 y continúa hasta el 16383. La parte RAM comienza en el 16384 y continúa hasta 32767 si se tienen 16K de RAM O 65535 si se tienen 48K. En un ordenador de 48K, todas las posiciones de memoria están asignadas a ROM o a RAM.

La instrucción POKE es fácil si se ha seguido la explicación del funcionamiento de la PEEK. POKE permite almacenar un byte en cualquier posición de memoria RAM. Haciendo esto se puede destrozar el programa si se direcciona una posición en la que resulta estar almacenado el programa. ¡Hay que ir con cuidado! La forma de POKE es

```
POKE dirección,byte
```

Por ejemplo, si (en una línea no numerada porque se desea que el ordenador realice la orden inmediatamente y no se quiere tener en memoria un programa que puede llegar a interferir) se teclea

```
POKE 17300,33
PRINT PEEK 17300
```

si el Spectrum está en funcionamiento se verá el 33 impreso

en la pantalla. El proceso que hemos realizado es almacenar una configuración de bits que representan al 33 en la posición de memoria cuya dirección es 17300 y luego imprimir su contenido. Probar lo mismo con diferentes bytes de datos para convencerse de que funciona. Si se intenta utilizar una POKE en direcciones que corresponde a ROM no se llegará muy lejos, por razones obvias.

## Aplicación de la PEEK para dibujar letras grandes

Una característica interesante y útil del área ROM de la memoria es el generador de caracteres. Si se utiliza una PRINT «A», de una forma u otra el Spectrum tiene que construir en la pantalla una configuración de puntos de papel y tinta que correspondan a la forma de la letra A. Para hacerlo busca la configuración en una tabla almacenada en la región ROM de la memoria. Esta tabla es el generador de caracteres y contiene una configuración de puntos para cada uno de los caracteres que el Spectrum puede imprimir. En el capítulo 2, cuando se trataron los gráficos definidos por el usuario vimos la forma de representar una configuración de puntos con una cifra de ceros y unos. Si se utilizan ocho posiciones de memoria por carácter podemos almacenar las ocho filas de ocho puntos necesarias para dibujar un carácter. Por ejemplo, la letra A sería:

| *configuración de puntos* | *número decimal* |
|---|---|
| 0 0 0 0 0 0 0 0 | 0 |
| 0 0 1 1 1 1 0 0 | 60 |
| 0 1 0 0 0 0 1 0 | 66 |
| 0 1 0 0 0 0 1 0 | 66 |
| 0 1 1 1 1 1 1 0 | 126 |
| 0 1 0 0 0 0 1 0 | 66 |
| 0 1 0 0 0 0 1 0 | 66 |
| 0 0 0 0 0 0 0 0 | 00 |

Si se encuentra que es difícil ver la A, intentarlo coloreando

todos los unos. La letra está rodeada de ceros para asegurar que al imprimir hay algún espacio alrededor de cada letra. La columna de números decimales corresponde a lo que se puede imprimir mediante la utilización de PEEK en las posiciones de memoria donde se almacena la forma de la letra A.

El problema de saber en qué parte de ROM se encuentra el generador de caracteres se soluciona fácilmente consultando el Capítulo 25 del manual del Spectrum, que revela que la primera dirección de la tabla está almacenada en las posiciones de memoria 23606 y 23607. En realidad la dirección correcta es memor de 256, por motivos que no importan ahora. Para encontrar el principio del generador de caracteres lo único que se debe hacer es escudriñar (PEEK) el contenido de las dos posiciones de memoria. Un número mayor de 255 se puede almacenar en dos posiciones de memoria pero se debe dividir en dos partes, cada una inferior a 255. (Más adelante se verá la forma de hacerlo). Para reconstruir el número es necesario utilizar la orden

```
PEEK a+256*PEEK(a+1)
```

donde a es la dirección de la primera (la inferior) de las dos posiciones de memoria. En el manual del Spectrum también se explica que la tabla de caracteres comienza con el carácter del espacio y continúa hasta el símbolo de derechos del autor CHR$(127). Sabiendo que cada definición de carácter ocupa ocho posiciones de memoria, se debe llegar a la conclusión de que la definición del carácter CHR$(i) está en la posición de memoria

```
principio +8*(i-32)
```

donde «principio» es el comienzo de la tabla de caracteres.

Sabiendo en qué lugar de la ROM está almacenada la tabla generadora de caracteres se puede escribir un programa que permita utilizar las configuraciones de puntos para imprimir o

trazar puntos con objeto de obtener caracteres más grandes. Para conseguirlo tenemos que solucionar un número determinado de problemas. Se puede utilizar PEEK para buscar el número almacenado en cualquier posición, pero es necesario conocer la configuración de ceros y unos. Dicho de otra forma, hay que buscar una manera de reducir un número a su secuencia de ceros y unos, lo cual no resulta nada difícil si se sabe aritmética y se entienden los números binarios. Para evitar vernos envueltos en demasiada teoría utilizaremos el siguiente programa, que proporciona una línea de ocho ceros, sin entrar en explicaciones.

```
10 LET principio=256+PEEK 23606+256*PEEK 23607
20 LET a=principio+8*(65-32)
30 LET r=PEEK a
40 FOR i=7 TO 0 STEP -1
50 LET b=r-2*INT(r/2)
60 LET r=INT(r/2)
70 PRINT AT 20,i;b
80 NEXT i
```

Como se puede ver, a contiene la dirección del comienzo de los 8 bytes que definen la letra A. La línea 30 escudriña (PEEK) la configuración de puntos de la primera fila de la letra A y la pone en la variable r. Cada vez que se ejecuta el bucle FOR se extrae un bit desde r y se imprime. La primera vez se extrae el bit más a la izquierda, después el siguiente por la derecha, y así sucesivamente hasta que se hayan impreso los 8 bits. El bucle FOR va de 7 a 0, debido a que el resultado se va a imprimir de izquierda a derecha; el índice i se utiliza en la PRINT AT 20, i. Se puede obtener toda la configuración contenida en r repitiendo el programa ocho veces, una por cada fila de la letra A.

```
10 LET principio=256+PEEK 23606+256*PEEK 23607
20 LET a=principio+8*(65-32)
30 FOR j=0 TO 7
```

```
40 LET r=PEEK(a+j)
50 FOR i=7 TO 0 STEP -1
60 LET b=r-2*INT(r/2)
70 LET r=INT(r/2)
80 PRINT AT 21,i;b
90 NEXT i
100 PRINT
110 NEXT j
```

Si se responde «y» a ¿«Scroll»?, se podrá ver la configuración de puntos de la letra A al ejecutar este programa. Ahora estamos casi salvados. Lo único que hay que hacer es añadir algunas instrucciones para imprimir un espacio en blanco cuando b sea 0 y un espacio negro cuando b sea 1; se obtendrá una letra A mayúscula en la pantalla. Añadir algún código para elegir la parte de la tabla correspondiente a cualquier cadena de letras y se tendrán mensajes largos moviéndose hacia arriba por la pantalla. Hacer la prueba con el siguiente programa:

```
10 INPUT a$
20 LET principio=256+PEEK 23606+256*PEEK 23607
30 FOR i=1 TO LEN a$
40 LET a=principio+8*((CODE a$(i))-32)
50 FOR j=0 TO 7
60 LET r=PEEK(a+j)
70 FOR k=7 TO 0 STEP -1
80 PRINT AT 21,k;CHR$(128+15*(r-2*INT(r/2)))
90 LET r=INT(r/2)
100 NEXT k
110 PRINT
120 NEXT j
130 NEXT i
```

Si se escribe un mensaje, saldrá en la pantalla como una secuencia de grandes letras en movimiento hacia la esquina superior izquierda de la pantalla.

s
p
e
c
t
r
u
m

La línea 40 elige la posición de cada letra en la tabla. La tabla comienza en la dirección almacenada en el «principio» y el carácter CHR$(32); un espacio en blanco es el primero. La función CODE es la contraria de la CHR$. Toma una letra y da su posición en el conjunto de caracteres. Una letra ocupa ocho posiciones de memoria, por lo que se debe multiplicar el código del carácter por ocho para obtener el lugar correcto en la tabla. La línea 80 utiliza un método idéntico al presentado en un programa anterior para decidir si es un cero o un uno, pero esta vez en lugar de imprimir 0 o uno, imprime CHR$(128+15*0), que es el carácter gráfico en blanco [8], o CHR$(128+15*1), que es el carácter gráfico completo [ˆ8]. Si se quiere repetir el mensaje continuamente añadir

```
130 GOTO 20
140 GOTO 20
```

Este programa se podría utilizar para añadir letras grandes en cualquier programa.

**La manipulación de las posiciones de memoria**

Como se dijo anteriormente, una posición de memoria sola puede tener un número comprendido entre 0 y 255, y tal como hemos visto, muchas veces se presenta la necesidad de almacenar números más grandes en un ordenador. Por ejemplo, la dirección de una posición de memoria del Spectrum, que puede estar comprendida entre 0 y 65534. Así, para almacenar una dirección sola se debe utilizar *dos* posiciones de memoria. Este funcionamiento está basado en el principio de los números binarios, pero desde el punto de vista de un programador de BASIC es una complicación innecesaria; es más fácil de entender y de recordar en términos de los números decimales que se utilizan en las órdenes de programación. El número más grande que una posición de memoria sola o byte puede tener es 255, de forma que si se utiliza una sola posición de memoria para contar, se puede comenzar desde 0 y contar

hasta que se llegue a 255. Si se intenta sumar uno a 255 no se puede guardar la respuesta, 256, en una posición de memoria sola pero se podría almacenar el uno en otra posición de memoria para indicar que se ha llegado una vez a 255. Entonces se podría continuar contando en la primera posición de memoria, como si nada hubiera sucedido (comenzando de nuevo desde 0) hasta llegar otra vez a 255. En realidad estaríamos utilizando la segunda posición de memoria como una cuenta del número de veces que se ha llegado a 256. La primera posición de memoria cuenta unidades y recibe el nombre de «byte menos significativo», y la segunda posición de memoria cuenta grupos de 256 unidades y recibe el nombre de «byte más significativo».

| primera posición de memoria Unidades | segunda posición de memoria Grupos de 256 |
|---|---|
| menos significativo | más significativo |

Ahora debe ser clara la forma de «reconstruir» una dirección a partir de dos posiciones de memoria. Como la primera cuenta unidades, podemos averiguar (PEEK) su valor sin problemas, pero la segunda cuenta grupos de 256, luego hay que multiplicar su valor por 256 antes de sumarla al primer valor. Pasando todo esto a BASIC, da:

```
PEEK m+256*PEEK(m+1)
```

He aquí la forma de averiguar qué hay en dos posiciones de memoria; m es la dirección de la primera. Resulta igual de fácil si se quiere dividir un número de forma que se pueda almacenar en dos posiciones de memoria; se divide por 256 y se ve cuántos grupos de 256 contiene, se almacena el resultado en el byte más significativo y luego se almacena el resto en el byte menos significativo. Es decir:

```
POKE m+1,INT(v/256)
POKE m,v-256*INT(v/256)
```

Este programa almacenará el número v en las dos posiciones de memoria m y m+1. Se ha utilizado ya el método para escudriñar (PEEK) dos posiciones de memoria. En el siguiente apartado veremos que algunas veces hay que meter datos (POKE) en las posiciones de memoria.

**Algunas posiciones de memoria útiles**

Si se mira el capítulo 25 del manual del Spectrum se verá un listado de las posiciones de memoria y para qué sirven. Muchas de estas posiciones sirven para propósitos que no tienen ningún interés para el programador de BASIC. No obstante, algunas de ellas pueden ser muy útiles. A continuación se ofrece una selección de las posiciones de memoria más útiles.

*Retardo de repetición automática*: El tiempo que se tiene que pulsar una tecla para que se repita automáticamente, puede ser seleccionado introduciendo (POKE) en la posición 23561 el retardo en cincuentavos de segundo.

*Velocidad de la repetición automática de la tecla*: La velocidad con la que la tecla se repite automáticamente se puede alterar introduciendo (POKE) la nueva velocidad en cincuentavos de segundo en 23562.

Estas dos posiciones de memoria son de gran utilidad si se quiere aumentar la sensibilidad del teclado mientras se ejecutan programas de juegos.

*Pitido de la pulsación de una tecla*: La duración del chasquido de la pulsación de una tecla se puede seleccionar introduciendo el valor adecuado en la posición de memoria 23609 mediante una POKE. Si este valor es suficientemente alto, el click que sigue a la pulsación de cada tecla se convierte en un pitido.

*Comienzo de los gráficos definidos por el usuario*: Las posiciones de memoria 23675 y 23676 contienen la dirección del comienzo del área RAM utilizada para almacenar las definiciones de los caracteres definidos por el usuario.

*Abscisa y ordenada de último punto trazado*: Pueden ser encontradas en 23677 y 23678 respectivamente. Estas posiciones se pueden escudriñar (PEEK) mediante un programa que descubra dónde se trazó el último punto, o atizar (POKE) para alterar la posición donde comienza DRAW.

*Supresión del corrimiento*: La posición de memoria 23692 cuenta las veces que la pantalla se ha movido hacia arriba una línea. Para decirlo con más exactitud, cuenta el número de corrimientos anteriores a la aparición en pantalla del mensaje de corrimiento («Scroll?»). Si este mensaje molesta, como en el caso del programa de las letras grandes mostrado anteriormente, puede ser suprimido introduciendo de vez en cuando en esta posición (mediante una POKE) un valor grande (por ejemplo, 255). Por ejemplo, para evitar que el mensaje de corrimiento «Scroll?» aparezca en el programa de las letras grandes, añadir

```
35 POKE 23692,255
```

**Resumen**

Este capítulo ha intentado ofrecer algunas ideas relativas al funcionamiento de las órdenes PEEK y POKE. El ejemplo de la utilización de PEEK para presentar imágenes de letras grandes es típico de la serie de cosas para las que se utilizan PEEK y POKE. Hay que tener en cuenta que aparte de saber cómo funciona PEEK, el ejemplo requiere un conocimiento sobre el ordenador; hay que conocer por ejemplo la existencia de un generador de caracteres, el lugar donde se ubica y el formato que tiene. Estas informaciones adicionales se confunden algunas veces con el funcionamiento y la forma de empleo

de las órdenes PEEK y POKE. Si se trabaja con un nuevo ordenador, la forma de empleo y el funcionamiento de PEEK y POKE continuarán siendo los mismos pero el programa de las letras grandes no funcionará. Puede que sea posible cambiarlo de forma que funcione si se sabe dónde está el generador de caracteres, etc. Ahora se puede comprender por qué no existe una respuesta a la pregunta «¿Para qué sirven PEEK y POKE?» si no se especifica de qué ordenador se habla.

# 8. UN SENTIDO DEL TIEMPO

Todos los ordenadores llevan incorporada alguna forma de percibir el tiempo. Algunos facilitan el acceso del programador a esta característica y otros lo hacen casi imposible. El Spectrum se encuentra en medio de estos dos polos, porque, por una parte, proporciona una orden de manipulación del tiempo, PAUSE, que permite el acceso a su reloj, mientras que, en la práctica, en la mayoría de los casos hay que utilizar la orden PEEK, y ocasionalmente la POKE. Antes de comenzar a analizar los métodos de manipulación del tiempo, veamos qué produce el paso del tiempo en el Spectrum.

## Un reloj interno

El microprocesador del Spectrum, el Z80, es responsable del mantenimiento de la imagen de la pantalla. Un aparato de televisión estándar presenta una imagen cada cincuentavo de segundo (un sesentavo de segundo en Estados Unidos); por lo tanto, el Spectrum tiene que interrumpir lo que está haciendo cada cincuentavo de segundo para presentar una imagen en la pantalla de televisión.

El «reloj» interno del Spectrum es realmente una característica muy práctica de la que se puede hacer un buen uso. Una forma de hacerlo es mediante la orden PAUSE. Hacer la prueba, por ejemplo, con el siguiente programa:

```
10 PRINT"tic"
20 PAUSE 50
30 PRINT"tac"
40 PAUSE 50
60 GOTO 10
```

Este programa imprimirá «tic tac» en la pantalla a intervalos aproximados de un segundo. El único problema con este programa es que a pesar de que cada PAUSE provoca una pausa durante un segundo (50 cuadros), el tiempo comprendido entre cada tic tac es mayor porque el ordenador dedica tiempo en las órdenes PRINT y GOTO.

## Bucles de retardo

Muchos ordenadores no tienen una orden PAUSE, por lo que resulta interesante conocer otra manera de crear un retardo determinado: el bucle de retardo. Un bucle de retardo es sencillamente un bucle FOR que tiene únicamente una fución: dejar pasar una cantidad de tiempo determinada. Por ejemplo:

```
10 LET t=200
20 PRINT "tic"
30 FOR i=1 TO t
40 NEXT i
50 PRINT "tac"
60 FOR i=1 TO t
70 NEXT i
80 GOTO 10
```

En este programa se incluyen dos bucles de retardo. Cada uno

de ellos proporciona un retardo ligeramente inferior a un segundo, lo que hace que el tiempo comprendido entre cada «tic» y «tac» sea aproximadamente de un segundo. Si se quiere comprobar la precisión del reloj y regularlo, la mejor forma es cronometrar una gran cantidad de «tic-tacs» y calcular cuánto dura cada uno. Si es menos de un segundo aumentar t y viceversa. No hay que olvidar que el tiempo de retardo depende del tipo de orden utilizada como bucle de retardo. Si se cambia FOR i=1 TO t por FOR i=1 TO 200, apenas cambiará el tiempo que tarda el bucle.

**El contador de cuadros**

Ya que el Spectrum puede hacer una PAUSE y presentar la imagen de un número determinado de cuadros de televisión, se puede deducir que, en algún sitio de su interior hay una posición de memoria que cuenta el número de cuadros que se han presentado en la pantalla. En realidad hay tres posiciones de memoria que siguen la cuenta del número de cuadros; estas tres posiciones reciben el nombre de CONTADOR DE CUADROS. El contador de cuadrados está en la dirección 23674, 23673 y 23672. La posición de memoria de la dirección 23672 cuenta el número de cuadros presentados desde que se conectó el ordenador. Como el número mayor que una posición de memoria puede contener es 255, el número de cuadros que la posición 23672 puede contar es limitado. Para solucionar este problema, la posición 23673 cuenta el número de veces que la posición inferior alcanza 255. Dicho de otra forma, la posición inferior cuenta cuadros y la posición superior cuenta grupos de 256 cuadros; es decir, el contador inferior va del 0 al 255 por cada vez que el contador superior cuenta una unidad. Es algo muy parecido a la esfera de un reloj tradicional; el contador inferior da una «vuelta» cada 256 y mueve entonces el contador superior una unidad. De la misma forma, cada vez que la dirección 23673 llega a 255, suma uno a la siguiente posición de memoria, la 23674. En resumen, la primera posición cuenta cada cincuentavo de se-

gundo, la segunda cada 256* cincuentavos de segundo, y la última cada 256*256* cincuentavos de segundo.

Para ver los contadores en funcionamiento, ejecutar:

```
10 PRINT PEEK(23672)+256*PEEK(23673)+65536*PEEK(23674)
20 GOTO 10
```

(Obsérvese que 256*256 es 65536). La diferencia entre los valores consecutivos es el número de cuadros que el Spectrum presenta en la pantalla entre cada impresión. Para ver este mismo número en segundos basta dividirlo por 50. Si además se quiere poner a cero el reloj al comienzo del programa, hay que introducir un cero en las tres posiciones (mediante una POKE). Por ejemplo:

```
10 LET p=23672
20 POKE p+2,0
30 POKE p+1,0
40 POKE p,0
50 PRINT PEEK(p)+256*PEEK(p+1)+65536*PEEK(p+2)
```

Después de ponerle a cero, para continuar contando añadir la línea 60.

```
60 GOTO 50
```

Conviene poner a cero en último lugar el contador que cambia más rápidamente. Igual que cuando se pone en hora un reloj; primero se pone la hora, luego los minutos, y finalmente los segundos.

**El reloj digital**

Hay muchas maneras de convertir el Spectrum en un reloj digital. Una de las más fáciles consiste en utilizar el programa anterior para ver el número de segundos transcurridos desde que se conectó el ordenador. Lo único que hay que hacer es

sumar la hora actual en segundos, pasar la respuesta a horas, minutos y segundos y presentar en la pantalla el resultado. Un método más interesante es el que se basa en el programa del «tic-tac». En lugar de utilizar el contador de cuadros para llevar la cuenta del tiempo ¿por qué no utilizarlo para señalar el paso de un segundo? Probar el siguiente programa:

```
10 LET p=23673
20 POKE p,0
30 IF PEEK p<>50 THEN GOTO 30
40 POKE p,50
50 PRINT"tic"
60 GOTO 30
```

Antes de que alguien llegue a la conclusión de que hay un error de imprenta, es obligado confesar que *este programa no funciona*. La razón de que no funcione es lo más interesante. Es difícil ver la razón de que el programa no funcione porque la idea en que se basa parece de lo más lógico. La línea 20 pone a 0 el contador de marcos inferior; la línea 30 comprueba si el valor del contador de cuadros inferior ha alcanzado cincuenta y en caso contrario lo vuelve a comprobar. En caso de que sean cincuenta sabremos que se han presentado en la pantalla cincuenta cuadros y que ha transcurrido un segundo. El contador vuelve a ponerse a 0 inmediatamente y comienza a contar el segundo siguiente, mientras se imprime «tic» en la pantalla. ¿Por qué no funciona? Ciertamente, hay tiempo suficiente de llegar a la instrucción IF antes de que la cuenta llegue a 0; ni siquiera el Spectrum necesita un segundo entero para ejecutar dos líneas de BASIC. El problema radica en la misma instrucción IF. ¡La instrucción IF tarda más de un cincuentavo de segundo en ser ejecutada! Esto significa que cuando el contador de cuadros cambia a 50 puede que el programa esté acabando de calcular el resultado de la última PEEK. Si se ejecuta con frecuencia el programa, es posible que salga algún «tic» en la pantalla; esto sucede cuando la instrucción IF lee fortuitamente el contador de marcos coincidiendo con el momento en que llega a 50.

Si se quiere utilizar el contador de cuadros como un temporizador interno, se debe elegir un intervalo de tiempo que sea largo comparado con el tiempo que el ordenador tarda en ejecutar una instrucción IF. El contador de cuadros superior cambia sólo una vez cada 256 cuadros, aproximadamente cada 5,12 segundos. Si se está dispuesto a tener un tic de reloj cada 5,12 segundos, se puede utilizar el contador de cuadros superior en el tipo de programa que no funciona con el contador de cuadros inferior. Probar con el siguiente programa:

```
10 LET s=0
20 LET h=20
30 LET m=39
40 LET p=23673
50 POKE p,0
60 LET a=PEEK p
70 LET b=a
80 LET a=PEEK p
90 IF a=b THEN GOTO 80
100 LET s=s+5,12
110 IF s<60 THEN GOTO 190
120 LET s=s-60
130 LET m=m+1
140 IF m<60 THEN GOTO 190
150 LET m=0
160 LET h=h+1
170 IF h<24 THEN GOTO 190
180 LET h=0
190 CLS
200 PRINT AT 0,0;h;AT 0,3;m;AT 0,6;INT s
210 GOTO 70
```

El programa lee el contador de cuadros superior en la línea 60 y vuelve a leerlo en la línea 80, a la espera de que la diferencia sea uno. Cuando esto sucede han pasado 5,12 segundos; la línea 100 puede actualizar el segundo contador y las líneas 110-200 presentan otra vez la hora en la pantalla. Este programa pone el reloj a las 20:39. Para poner otra hora al-

terar los valores de los segundos, horas y minutos de las líneas 10 a 30.

Es obvio que hay mucho por hacer si se quiere perfeccionar este programa: por ejemplo, puede ser interesante añadirle la presentación visual de letras grandes, mostrada en el capítulo siete, o la ayuda de un reloj despertador. Pero en el caso de que se decida incorporar un click utilizando la orden BEEP para imitar el sonido del tic-tac, hay que tener presente que el contador de cuadros deja de contar durante la ejecución de la orden BEEP.

## Un reloj de ajedrez

Una aplicación sencilla del contador de cuadros es un reloj de ajedrez:

```
10 LET tw=0
20 LET tb=0
30 LET g=0
40 LET p=23672
50 PRINT "Pulsar cualquier tecla para iniciar el juego"
60 IF INKEY$="" THEN GOTO 60
70 CLS
80 POKE p+2,0:POKE p+1,0:POKE p,0
90 LET t=(PEEK(p)+256*PEEK(p+1)+65536*PEEK(p+2))/50
100 IF g=1 THEN PRINT AT 5,15;"BLACK";INT ((t+tb)/60);".";
    INT((t+tb)-INT((t+tb)/60)*60);" "
110 IF g=0 THEN PRINT AT 5,0;"WHITE";INT((t+tw)/60);".";
    INT((t+tw)-INT((t+tw)/60)*60);" "
120 IF INKEY$=" " THEN GOTO 90
130 BEEP .1,0
140 IF g=0 THEN LET tw=tw+t
150 IF g=1 THEN LET tb=tb+t
160 LET g=NOT g
170 GOTO 80
```

En realidad no hay nada nuevo en este programa. A estas alturas se debe estar en condiciones de incluir algunas técnicas de los capítulos anteriores. El tiempo del movimiento completo es guardado en tw para las blancas y en tb para las negras. La variable g es 1 si están jugando las negras y 0 si son las blancas. Pulsando cualquier tecla cambia el jugador (línea 120). Las líneas 100 y 110 suman el tiempo t transcurrido desde el último cambio al tiempo de juego total jugado por las negras y las blancas respectivamente, y la línea 80 pone a cero el reloj. La única limitación de este reloj de ajedrez consiste en que cada movimiento debe tardar menos de cuatro días, pues de lo contrario el contador llega a su cuenta máxima y comienza a contar de nuevo.

## Un juego de tiempos de reacción

Utilizando el contador de cuadros inferior es posible cronometrar situaciones con una precisión de un cincuentavo de segundo. Esta característica hace factible escribir un programa de tiempos de reacción. Es muy importante comprender que, debido a la lentitud del BASIC del Spectrum, la exactitud en la medición de los tiempos de reacción es inferior a un cincuentavo de segundo, lo cual no es suficiente para hacer algo en serio, pero resulta divertido.

```
10 PRINT "preparado"
20 FOR i=0 TO RND*100+50
30 NEXT i
40 LET p=23672
50 POKE p+1,0
60 POKE p,0
70 PRINT "ya"
80 IF INKEY$=" " THEN GOTO 80
90 LET t=PEEK(p)+PEEK(p+1)*256
100 PRINT "tiempo de reacción=";t/50;"seg"
110 GOTO 10
```

Después de un retardo aleatorio sale impresa la palabra «ya». La pulsación de cualquier tecla produce la lectura y la impresión en la pantalla del tiempo. Se puede hacer que este programa sea más interesante y preciso tomando la media de 10 mediciones de tiempos de reacción. Una vez se ha comprendido el programa anterior, ejecutar y probar el siguiente:

```
10 LET s=0
20 FOR j=1 TO 10
30 CLS
40 PRINT "preparado"
50 FOR i=0 TO RND*50+40
60 NEXT i
70 LET p=23672
80 POKE p+1,0
90 POKE p,0
100 PRINT "ya"
110 IF INKEY$=" " THEN GOTO 110
120 LET t=PEEK(p)+PEEK(p+1)*256
130 LET t=t/50
140 LET s=s+t
150 NEXT j
160 CLS
170 PRINT "Tu media es";s/i
180 IF s/i>.08THEN PRINT "Lento"
190 IF s/i>.05 AND s/i<=.08 THEN PRINT "No está mal"
200 IF s/i<.05 THEN PRINT "Bien"
210 IF s/i<.02 THEN PRINT "Muy rápido"
```

Las líneas adicionales situadas al final del programa calculan la puntuación sobre diez intentos e imprimen el mensaje apropiado. Esta rutina puede ser utilizada como base para una variedad de juegos, pero es importante no olvidar que la precisión de este tipo de temporización es muy limitada.

# 9. CADENAS Y PALABRAS

Es muy fácil caer en el error de pensar que los ordenadores tratan principalmente con números y con cálculos complicados y tediosos. Como se habrá deducido ya, nada más lejos de la realidad. Concretamente, el Spectrum es muy bueno en la manipulación de textos. Hay, sin embargo, una serie de problemas con la utilización de textos para programas de juegos, problemas que todos los ordenadores comparten. Se han solucionado algunos de estos problemas, pero otros nos conducen al límite de nuestro conocimiento sobre los ordenadores, y nos introducen en el campo de la inteligencia artificial.

## Cadenas y cosas

Antes de iniciar el tema de la forma de empleo de las cadenas, puede ser conveniente incluir un pequeño resumen sobre la forma en que el Spectrum maneja las cadenas. El Spectrum diferencia las variables de cadena de las demás mediante un signo $ puesto detrás de un nombre de variable. Por ejemplo:

```
10 LET a$="Nombre"
```

La cadena puede ser de cualquier longitud siempre y cuando quepa en la memoria. Las cadenas se pueden manipular de tres formas.

Según el primer método, se pueden unir cadenas mediante la operación+. Por ejemplo:

```
10 LET a$="Primer nombre"
20 LET b$="Ultimo nombre"
30 LET a$=a$+b$
40 PRINT a$
```

Este programa toma la cadena «Primer nombre» y la cadena «Ultimo nombre» y las junta para poner «Primer nombre Ultimo nombre» en la variable a$.

Según el segundo método, se puede tomar cualquier parte de una cadena mediante la notación de partición. Por ejemplo:

```
10 LET a$="abcdefg"
20 PRINT a$(2 TO 5)
```

imprimirá la cadena abcdefg desde la segunda letra a la quinta, por ejemplo bcde. Se puede utilizar la notación a$ (comenzar TO acabar) cuando «comenzar» y «acabar» son sustituidos por números que nos dicen que la cadena contenida en a$ va desde una letra, incluyendo la letra «comenzar» hasta otra letra, incluyendo la letra «finalizar». El Spectrum también permite ciertas formas cortas de la notación de partición.

```
a$ (TO n) = a$ (1 TO n)
a$ (n)   = a$ (n TO n) la letra enésima
a$ (n TO) = a$ (n TO LEN (a$))
a$ (TO) = a$ (1 TO LEN (a$)) la cadena completa
```

El tecer método de manipular cadenas es realmente ingenioso.

Se puede cambiar la parte de una cadena especificada mediante la notación de partición. Por ejemplo, el programa

```
10 LET a$="abcdefg"
20 LET a$(2 TO 3)="12345"
30 PRINT a$
```

cambiará la cadena abcdefg por a12defg. Dicho de otra forma, el separador especifica la parte de la cadena que se ha de cambiar: la segunda letra y la tercera. No importa si la cadena situada a la derecha del signo igual es mayor que la parte que se va a cambiar: utiliza el número correcto de caracteres comenzando desde la izquierda. En el ejemplo presentado sólo se utiliza «12» aunque la cadena es «12345».

Como ejemplo sencillo del manejo de las cadenas, ejecutar el siguiente programa:

```
10 INPUT a$
20 LET b$=""
30 FOR I=LEN a$ TO 1 STEP -1
40 LET b$=b$+a$(I)
50 NEXT I
60 PRINT b$
```

Este programa lee cualquier cadena y la invierte. Escribir la siguiente frase «oditrevid res edeup amargorp etse» para descubrir lo que dice. Observar que este programa «desmembra» la entrada en letras individuales utilizando la notación de partición y la recompone en el orden inverso, mediante la línea 40. Se podría utilizar este programa para enviar mensajes secretos o simplemente para aprender a hablar al revés.

### Palabras aleatorias

En el capítulo 2 se vio la forma de empleo de los números aleatorios y en el capítulo 3 la forma de convertir números

aleatorios en gráficos aleatorios. Utilizando las mismas técnicas se pueden generar caracteres aleatorios. Hacer la prueba con el siguiente programa:

```
10 PRINT CHR$(32+INT(RND*224));
20 GOTO 10
```

La pantalla se llenará de caracteres aleatorios, más o menos así:

```
  MOVE /JM?SQR OASN PAUSE RNDM THEN DPOINT RANDOMIZE LN
% FORMAT ANF <= SAVE Q STOP X1 STEP HO FN SQR RETURN COS
K] INT CUTAB awi DI M Or   PEEK . OR WT STOP I<> ASN I DATA LN
4 OVER NEXT IF + GO SUB RESTORE DRAW FLASH IN geT ASN DR
AW OUT F ERASE AND MERGE CODE 60 COS   : AND FOR 8>D1 LIST
LN AfLN /+ THEN ty PAUSE CODE CLEAR Y<U J <> CHR$ NF DRAW
BEEP CUW   TO BRIGHT CLEAR >9   0   INVERSE qM1 NEW UI STOP
CLEAR X CAT TAB ULN RAND OMIZE <> PINE INVERSE   STOP TABS
DEF FN IYwt I NEW   LN PNINKEY$9xN   NEW GE NEXT NEXT FOR
CLEAR ;Y6T AN OPEN # PRINT i STEP >= LET COPY LOAD STAB
AINT DATA NOT STOP   4USR BEEP V OPEN #BHC DEF FN   BIN Dg
PRINT   7 INVERSE      RETURN   RUTPIy LET bU5*ATTR BIN I
POKE ZCQ   GI" CONTINUE : CONTINUE CLS
```

El siguiente paso es intentar generar palabras al azar. Si se observa la pantalla llena de caracteres aleatorios, que generó el programa anterior, se verán algunas palabras como GOTO, RND, COPY, etc. Son, simplemente, las palabras reservadas del BASIC del Spectrum impresas en el teclado. Para el Spectrum estas palabras son caracteres individuales. Se introducen por el teclado mediante la pulsación de una tecla y quedan almacenadas en una posición de memoria. La única diferencia radica en que, cuando son impresas, las palabras reservadas se multiplican en muchas letras. Si no es mediante las palabras reservadas, es *muy* difícil generar palabras al azar en un ordenador. Hay que tener mucha suerte para obtener palabras largas mediante la generación de letras al azar (excluyendo las palabras reservadas y los gráficos). Hacer la prueba con:

```
10 PRINT CHR$(INT(RND*26)+65);
20 GOTO 10
```

```
N C X N Z B S B P M R P H V A Q T I W X O R Z K X F Q L I P L D
L K O Z Y F N W P R F M T Q T H K B U K Q L W G R V S L T A R W
E Z W W K T G R E E B C L Y X H N G R Q S E Y D V Y Y V O I Z X
M R G G C F N J G K V P Y M X S J S I T G N O B T Z L J V B E R
Q F K I B V Q E V I B A L P E J U A W A F P I I T E I G A V F A
Z R J N F T R B B S I X W M E X G C W T E Q O W F V H Y H Z X J
C R B D A I F H T F D L T E L T L W U N Y W P W D O J D Y X R M
U H K L A K M L A U M C I V S J L A J L Y Y Q R T L N B J W D U
I J G S J P R P L I K R J W V C U B X F Q I N Q R E I Y R Z I H
O X K A N X H M E B W H W D Q R G Y P K A Q J C T J G A Z Y U P
H V S C A Z H J W X S L U Z N R V J P X R L P Z J N F Z D H J D
A F T M M B Y H B C U V W I B A I X X W I K A Z F Y C Q N F Q H
L Z X T A D E V Q I I T B Q Z N J O E Z R S H M N R J C D M P P
N Q S O N L L W Q M Ñ W J J Y J L N U V A C Y H F W O K M T G Y
P F D D M Z V O M S S B G S M A M W Q N O J L N T S I U L V P Z
O K U N N R Q N E O Q R P O Y N Z M V K E A J N G B Q J M Z J C
B D N P U X E S M T D M V Y Y Y J H T W O D D H T C M I L B M E
K Q M Y X L G A G G E J V K D E G Z D E O E G G R A P A H K C D
O D F T P W T R K C L D A G P Q A E V K G I M N Y U E F O Q R M
X R A N V Z J E J D U A C Y K X D B W C Q G X G M W K M R U D I
B Y A U X E P X E Y N Q S U Z R S H J V K W Q N Q O Q W K Z H O
T M G U E G X V E G U A F M V Y S P U H V Q V C E Z H Q A K B D
```

Se pueden descifrar algunas palabras de tres o cuatro letras que no son muchas si se compara con la cantidad de series de letras sin significado que hay.

La dificultad de la generación de palabras al azar limita profundamente el tipo de juegos de palabras que un ordenador puede permitir. Por ejemplo, si se quiere programar un juego de adivinar números, el ordenador puede generar un número aleatorio y uno puede intentar adivinar cuál es. Pero en el caso de un juego de adivinar palabras, el ordenador no ofrece ninguna posibilidad de generar palabras al azar; lo mejor que se puede hacer es introducir un listado de palabras al principio del juego y programar el ordenador para que elija una palabra al azar. Se puede utilizar este método para programar juegos de palabras, si se puede conseguir que sea otra persona la que introduzca la lista de palabras posibles o si el listado es lo bastante largo como para que resulte imposible recordar todas las palabras.

### El juego del ahorcado

Comenzemos escribiendo un programa de una versión simple del juego del ahorcado. El siguiente programa cuenta con

que alguna persona ajena al juego introduzca un listado de palabras cuando el jugador no esté mirando.

```
10 LET w$="<"
20 FOR i=1 TO 4
30 INPUT a$
40 LET w$=w$+a$+"<"
50 NEXT i
60 CLS
70 PRINT "EL AHORCADO"
80 LETr=INT(RND*(LEN w$-1)+1)
90 IF w$(r)="<" THEN GOTO 120
100 LET r=r-1
110 GOTO 90
120 LET a$=" "
130 LET w$=w$(1 TO r-1)+w$(r+1 TO)
140 IF w$(r)="<" THEN GOTO 170
150 LET a$=a$+w$(r)
160 GOTO 130
170 FOR i=1 TO LEN a$
180 PRINT "*";
190 NEXT i
200 LET h=0
210 PRINT AT 3,0;"Adivinar una letra"
220 INPUT b$
230 LET k=0
240 FOR i=1 TO LEN a$
250 IF b$(1)=a$(i) THEN LET k=i
260 NEXT i
270 IF k=0 THEN GOTO 210
280 LET a$(k)="*"
290 LET h=h+1
300 PRINT AT 1,k-1;b$(1)
310 IF h<>LEN a$ THEN GOTO 210
320 PRINT AT 4,0;"Well done"
330 PAUSE 100
340 IF LEN w$>1 THEN GOTO 60
```

El programa comienza (líneas 10 a 60) solicitando que alguien introduzca cuatro palabras. A medida que son escritas, las palabras son añadidas al listado de palabras almacenado en w$. Cada palabra del listado se separa mediante «<». Si se quiere verlo, añadir 45 PRINT w$ al programa. Después de borrar la pantalla, el programa pasa al juego del ahorcado propiamente dicho. Lo primero que hay que hacer es elegir al azar una palabra del listado de palabras. Las líneas 80 a 160 realizan esta función. En primer lugar, la línea 80 genera un número aleatorio menor que el número de caracteres del listado de palabras (w$). El número aleatorio puede ser considerado como la forma de «señalar» la palabra que se ha seleccionado. Luego se debe pasar la palabra elegida a otra variable de cadena (a$) y suprimirla del listado de palabras para que no pueda ser escogida de nuevo. Esto se hace devolviendo el puntero r al primer «<» y luego pasando todo lo que hay hasta el siguiente «<» a w$, mediante las líneas 90 a 160. Después de seleccionar al azar la palabra a acertar, el programa continúa imprimiendo un asterisco «*» por cada letra de la palabra (líneas 170 a 190). Luego el programa espera a que se introduzca una adivinanza, en la línea 220. El bucle FOR compuesto por las líneas 240 a 260 compara la adivinanza con la palabra a acertar, y si encuentra alguna pareja de letras coincidentes, almacena la posición de la coincidencia en la variable k. Las líneas 270 a 310 imprimen la letra correcta en la posición correcta de la palabra, y sustituyen la letra acertada de la palabra a acertar por un «*» (línea 280). Al sustituir la letra con una «*» se evita que sea cogida de nuevo como respuesta correcta en posteriores adivinanzas. Si el número de adivinanzas correctas es igual al número de letras de la palabra a acertar, se habrá adivinado la palabra completa: la línea 320 imprimirá un mensaje de felicitación y el programa cogerá al azar la siguiente palabra, en caso de que haya otra.

Este programa pone en práctica diversos métodos, todos interesantes, por lo que merece la pena estudiarlo. Si se tiene la suficiente paciencia se podría aumentar el número de palabras posibles hasta alrededor de 100, y luego uno mismo po-

dría introducir en el ordenador el listado de las palabras, ya que es casi imposible acordarse de las cien palabras escritas después de varias veces de jugar al ahorcado. También se puede intentar añadir algunos gráficos e idear un sistema de marcador del número de intentos utilizados para llegar a la respuesta correcta.

## Códigos y mensajes cifrados

Los ordenadores, adecuados para el tratamiento de números y textos, constituyen una herramienta obvia para cualquier persona interesada en los códigos y en los mensajes cifrados. Durante la Segunda Guerra Mundial, los ordenadores pusieron al descubierto una gran cantidad de información «violando» mensajes cifrados. De hecho, no es posible utilizar el Spectrum para romper códigos pero sí puede constituir una buena máquina de codificar y descodificar utilizando las prestaciones de las funciones RND y RAND, descritas en el capítulo 3.

```
10 PRINT "CODER"
20 PRINT "¿Cual es la clave?"
30 INPUT k
40 RAND k
50 PRINT "Descifrar o cifrar (0/1)"
60 INPUT d
70 IF d=0 THEN LET d=-1
80 PRINT "Escribir mensaje"
90 INPUT a$
100 FOR i=1 TO LEN a$
110 LET a=CODE a$(i)-64
120 IF a$(i)=" " THEN LET a=0
130 LET a=a + d*INT(RND*59)
140 LET a=ABS(a-INT(a/59)*59)
150 IF a=0 THEN LET a=-32
160 PRINT CHR$(a+64);
170 NEXT i
```

Para probar el programa, intentar descifrar el siguiente mensaje:

```
WqMhBhVlUfx ZeILi
```

Ejecutar el programa y responder 1983 a la pregunta «¿Cuál es la clave?» Luego responder 0 a la pregunta «descrifrar / cifrar» e introducir la cadena de códigos mostrada últimamente: En la pantalla saldrá el mensaje descifrado.

Este programa utiliza el hecho de que mediante el uso de la función RAND se puede conseguir una secuencia específica de números aleatorios. Lo único que hay que hacer para obtener exactamente la misma secuencia de números es usar el mismo valor al definir RAND. No hay que olvidar que RAND 0 tiene un significado especial (selecciona el comienzo del generador de números aleatorios en función del tiempo transcurrido desde que se conectó el ordenador), que nos podría proporcionar una clave desconocida: una clave que no se podría repetir con vistas a su desciframiento posterior. Cero es por lo tanto el valor de uno que nunca se debe usar en este programa. El programa (línea 30) solicita la introducción del valor de la «clave» elegida, y (línea 40) pone en marcha el generador de números aleatorios. Para descifrar un mensaje hay que usar obligatoriamente el mismo valor clave, por lo que si no se sabe qué clave fue usada para codificar un mensaje, el mensaje no se puede descifrar. El mensaje introducido en la línea 90 es desmembrado en letras, y cada letra convertida en un número mediante la función CODE. Restando 64 del valor CODE evitamos la obtención de caracteres gráficos en el resultado, ya que los caracteres gráficos pueden ser difíciles de escribir y enviar a alguien. En la línea 120 se da al espacio en blanco el valor 0. El resto de la codificación consiste en sumar un número aleatorio comprendido entre 0 y 59 y calcular el resto cuando se divide por 59. Cuando se divide por 59 el resto se encuentra en la misma gama que los códigos de caracteres con los que comenzamos, del 0 al 58. Para imprimir los caracteres resultantes se utiliza CHR$(a+64), sin

olvidar tampoco ahora corregir el carácter del espacio (línea 150). Para descodificar el mensaje, se resta el número aleatorio entre el 0 y el 59 del código para volverlo a convertir en su valor original anterior a la codificación. La variable d se pone a −1 para descifrar y a 1 para cifrar.

Este programa de codificación, aunque pequeño, es suficientemente bueno para producir códigos difíciles de violar. Si no se tiene la clave, es virtualmente imposible leer un mensaje específicado por el Spectrum, ya que los caracteres no están clasificados en grupos separados por espacios y un mismo carácter puede representar a diferentes caracteres en diferentes partes del mensaje.

**Números en lugar de palabras - Un juego de adivinanza de números**

Si se quiere jugar a las adivinanzas de números con un juego de los que implican adivinar dígitos individuales, se necesitará disponer de alguna forma de emparejar la adivinanza con el número a adivinar, y se necesitará saber generar al azar un número de una cantidad determinada de dígitos. Para conseguirlo se necesitan cadenas con objeto de manipular los dígitos individuales. El juego de la adivinanza de números que se muestra a continuación es muy conocido: El ordenador saca al azar un número de cuatro dígitos no repetidos, el jugador conjetura un número de cuatro dígitos, y el ordenador dice cuántos dígitos del número elegido están:

(a) en el número a acertar,
(b) en el mismo lugar del número a acertar.

Los dígitos que están en el mismo lugar que en el número a acertar reciben el nombre de aciertos completos y los dígitos que están en diferentes sitios reciben el nombre de aciertos parciales. Por ejemplo, si el ordenador ha elegido 1234 como el número a acertar y el jugador ha conjeturado 2035, se ten-

drá un acierto completo, el tres, y un acierto parcial, el dos. El motivo de no permitir que los números a acertar tengan dígitos repetidos es evitar cualquier dificultad a la hora de contar el número de aciertos parciales.

```
10 LET a$="":LET g=0
20 RAND 0
30 FOR i=1 TO 4
40 LET b$=STR$ INT(RND*10)
50 FOR j=1 TO LEN a$
60 IF a$(j)=b$ THEN GOTO 40
70 NEXT j
80 LET a$=a$+b$
90 NEXT i
100 INPUT b$
110 IF LEN b$<>4 THEN BEEP .5,0 :GOTO 100
120 LET p=0: LET h=0
130 LET g=g+1
140 FOR i=1 TO 4
150 IF a$(i)=b$(i) THEN LET p=p+1
160 FOR j=1 TO 4
170 IF a$(j)=b$(i) THEN LET h=h+1
180 NEXT j
190 NEXT i
200 LET h=h-p
210 PRINT "Adivinar";b$
220 PRINT "Aciertos completos=";p,"aciertos parciales=";h
230 IF a$<>b$ THEN GOTO 100
240 PRINT "Correcto en ";g;" adivinanzas"
```

Las líneas 30 a 90 generan el número de cuatro dígitos aleatorios. La línea 40 genera un dígito aleatorio (0 a 9) en forma de cadena y las líneas 50 a 70 le comparan con los dígitos que se han presentado con anterioridad. Si éste ya está presente se vuelve atrás y se genera otro. En la línea 80, si este dígito no está presente se suma al número. En la línea 100 el programa solicita una conjetura, pero si no tiene cuatro números es rechazada mediante una BEEP y el programa continúa es-

perando una entrada correcta. Las líneas 140 a 190 comprueban las adivinanzas correctas contabilizando los aciertos completos y los aciertos parciales. La línea 150 comprueba los aciertos totales, lo cual resulta fácil comparando ambos números dígito por dígito, y sumando uno a p por cada pareja. Para comprobar los aciertos parciales se necesita un bucle FOR adicional (líneas 160 a 180), ya que resulta ligeramente más complicado. Cada dígito de la adivinanza se compara con *cada otro* del número a acertar y se suma uno a h por cada pareja de números idénticos. La línea 200 cuenta los aciertos parciales incluyendo también los dígitos que están en la posición correcta, por lo que se debe restar p para obtener una cuenta precisa de los aciertos parciales. La línea 220 imprime el número de aciertos completos y aciertos parciales y la 230 comprueba si se ha acertado el número. En la siguiente muestra de la salida, el número del ordenador era el 8340 y el jugador tuvo que intentarlo cinco veces para conseguir adivinarlo.

```
Adivinar         1234
Aciertos          = 0            aciertos          =     2
completos        0987            parciales
Adivinar          = 0                              =     2
Aciertos         1279            aciertos
completos         = 0            parciales         =     0
Adivinar         3460
Aciertos          = 1            aciertos          =     3
completos        8340            parciales
Adivinar          = 4                              =     0
Aciertos                         aciertos
completos                        parciales
Adivinar
Aciertos                         aciertos
completos                        parciales
Correcto en 5 intentos
```

En el programa no se incluyen indicaciones ni mensajes, y se puede perfeccionar ámpliamente añadiéndole más sonido y color, al gusto de cada uno. Pero incluso en su forma básica, este juego puede crear adicción. ¡Peligro!

# 10. GRAFICOS AVANZADOS

Sucede a veces que es más fácil y más práctico escribir un programa empleando una combinación de gráficos de alta resolución y de gráficos de baja resolución, pero esto acarrea siempre la dificultad de acoplar los dos sistemas de coordenadas. El siguiente programa es un ejemplo de este tipo de situaciones.

## Mezcla de resoluciones - El juego de las cargas de profundidad

Este juego consiste en animar dos navíos (uno es un navío de superficie y el otro es un submarino) y permitir al primero que «lance» una carga de profundidad sobre el otro. Resulta fácil presentar en la pantalla los navíos mediante gráficos de baja resolución, pero las cargas de profundidad se representan mejor mediante gráficos de alta resolución. Si se ha seguido el análisis del Capítulo 6, se estará capacitado para escribir este programa. Para animar los dos navíos podríamos utilizar el método general de imprimir una figura en la primera posición, borrarla y volverla a imprimir un poco más adelante.

Sin embargo, ya que los navíos se van a mover horizontalmente en una línea recta, se puede utilizar un truco para simplificar la programación. Ejecutar el siguiente programa:

```
10 FOR x=0 TO 30
20 PRINT AT 3,x;"[8]*";
30 NEXT x
```

Se verá un asterisco moverse de izquierda a derecha por la parte superior de la pantalla. Adviértase que este programa mezcla la reimpresión del asterisco con el borrado de la posición anterior mediante la inclusión de un espacio en blanco en el «objeto» en movimiento. Normalmente siempre es posible utilizar esta técnica. Lo único que hay que hacer es asegurarse de que la posición donde se imprime el «objeto» tenga alrededor suficientes espacios en blanco para «borrar» la versión anterior. En la práctica resulta demasiado difícil, a menos que el objeto se mueva en línea recta. (Otro ejemplo de este método es el bate del juego de «Squash»).

Otros problemas específicos del juego de las cargas de profundidad son decidir cuando se acierta al submarino y generar una «explosión» adecuada para suprimirlo de la pantalla, pero todo lo que esté relacionado con las cargas de profundidad resulta complicado por el hecho de que se generan utilizando gráficos de alta resolución mientras que el navío y el submarino se generan utilizando gráficos de baja resolución. A continuación se presenta el listado del programa completo. (Tener presente que los caracteres gráficos están señalados por corchetes que encierran la letra que hay que pulsar para generarlos).

```
10 LET mc=0
20 LET hc=0
30 LET f=0
40 LET h=0
50 LET xs=0
60 GOSUB 600
```

```
70 LET x=0
80 LET ys=INT(RND*5)+15
90 LET y=2
100 GOSUB 200
110 PRINT AT 0,0;"Aciertos=";hc;TAB(10);"fallos=";mc
120 IF f=0 THEN LET xd=8*x:LET yd=140
130 REM es el indicador de lanzamiento
140 IF INKEY$="f" AND f=0 THEN LET f=1
150 IF f=1 THEN GOSUB 300
160 GOSUB 400
170 IF h=0 THEN GOTO 100
180 GOSUB 500
190 GOTO 30

200 PRINT AT y,x;"[8][^3][^8][^3][^3]";
210 LET x=x+1
220 IF x>25 THEN LET x=0
230 IF x=0 THEN PRINT AT y, 26;"[8][8][8][8]";
240 RETURN

300 PLOT INVERSE 1,xd,yd
310 LET yd=yd-4
320 PLOT xd,yd
330 REM hit?
340 IF yd<16 THEN LET f=0:LET mc=mc+1
350 IF f=0 THEN PLOT INVERSE 1,xd,yd
360 IF ABS(xd-8*xs-24)>24 THEN RETURN
370 IF ABS(yd-175+8*ys)>8 THEN RETURN
380 LET h=1
390 RETURN

400 PRINT AT ys,xs;"[8][^2][^3][^3]";
410 LET xs=xs+RND
420 IF xs>28 THEN LET xs=0
430 IF xs=0 THEN PRINT AT ys,28;"[8][8][8][8]";
440 RETURN

500 LET hc=hc+1
```

```
510 FOR i=1 TO 20
520 PRINT AT ys,xs;"[6][^4][4][5]";
530 PRINT AT ys,xs;"[8][8][8][8]";
540 NEXT i
550 RETURN

600 CLS
610 FOR i=0 TO 31
620 PRINT AT 3,i;"[^3]";
630 PRINT AT 21,i;"[3]";
640 NEXT i
650 RETURN
```

Aciertos = 1 Fallos = 1

El programa comienza utilizando la subrutina 600 para inicializar variables y dibujar el mar y el fondo del mar. La línea 80 selecciona al azar la profundidad del submarino. La subrutina 200 traza el navío de superficie en la posición x,y y la subrutina 400 traza el submarino en la posición xs,ys. El submarino se mueve en la misma dirección que el navío de superficie y el espacio que recorre es aleatorio (línea 410). La carga de profundidad se lanza pulsando la tecla «F»; la línea 140 detecta esta acción mediante INKEY$. Una vez lanzada

la carga de profundidad, la variable f es puesta a 1 y la posición de la carga de profundidad es almacenada en xd,yd. La subrutina 300 se encarga del seguimiento de la posición de la carga de profundidad y de mantenerla en movimiento. Adviértase que para generar las cargas de profundidad se utiliza la orden PLOT y que para generar los navíos se utiliza la orden PRINT AT, luego hay un problema en la comparación de las coordenadas; PLOT trabaja con una pantalla de 256 por 176 y PRINT AT trabaja con una pantalla de 32 por 22. Lógicamente, xd y yd están determinadas tomando como referencia la parte inferior de la pantalla pero y está determinada a partir de la parte superior de la pantalla. Todo esto hace que comenzar el lanzamiento de una carga de profundidad y decidir si ha alcanzado al submarino sea más difícil de lo que cabría esperar. Las líneas 120 y 130 mantienen xd y yd en la misma posición que el navío de superficie; de esta forma, cuando se lanza la carga de profundidad, ésta comienza a caer desde la posición actual del navío de superficie. Las líneas 360 y 370 supervisan si la carga de profundidad ha acertado al submarino. Hay que tener en cuenta que para comparar las dos coordenadas x (abscisas) basta multiplicar por ocho pero para comparar las dos coordenadas y (ordenadas) hay que restar 8*ys de 175. El resto del programa es bastante fácil, pero obsérvese de todas formas cómo «destruyen» las líneas 510 a 540 el submarino. Esto puede resultar útil en otros juegos.

### La composición de la pantalla

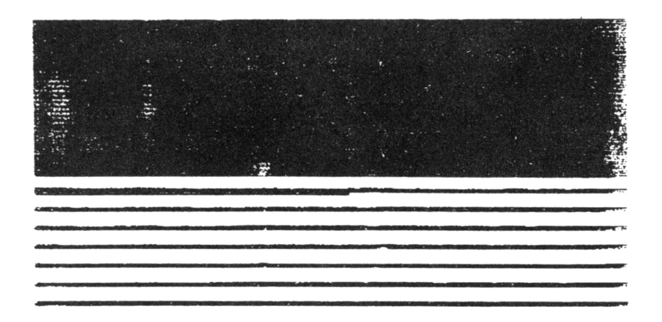
Todo lo que sale en la pantalla de televisión es almacenado en la memoria del Spectrum. Saber dónde y cómo almacenar las imágenes puede resultar muy útil e incluso puede sugerir nuevas formas de utilizar los gráficos. Tal como se indicó en el Capítulo 2, la información sobre lo que debe salir en la pantalla se divide en dos partes: la memoria de los puntos de papel y tinta y la memoria de los atributos. La memoria de los puntos de papel y tinta almacena información relativa a si los puntos de las 256 × 176 posiciones de la red de la pantalla

son puntos de tinta o de papel. La memoria de los atributos se utiliza para almacenar el color actual de la tinta, el color del papel y el estado de brillo y de intermitencia de la posición del carácter. Para facilitar las cosas las estudiaremos individualmente.

El área de los puntos de papel y tinta de la memoria comienza en la posición de memoria 16384 y va hasta la 22527. Una posición de memoria puede almacenar el estado de un total de ocho puntos. La correspondencia entre las posiciones de memoria y las posiciones de la pantalla es particularmente complicada; comenzando por la fila superior de puntos, el primer grupo de ocho (de la izquierda) se almacena en la primera posición de memoria, el siguiente grupo de ocho se almacena en la siguiente, y así sucesivamente hasta el final de la fila. Es decir, las 32 primeras posiciones de memoria contienen la fila superior de puntos de la pantalla. Obsérvese que estas 32 posiciones de memoria también corresponden a las 32 posiciones de impresión del carácter. Dicho de otra forma, una posición de memoria almacena la fila superior de ocho puntos de una posición de carácter. Si todo fuera así de sencillo, las siguientes 32 posiciones de memoria podrían contener la segunda fila de puntos, pero desafortunadamente no es este el caso. En realidad es la primera fila de puntos, que compone la segunda línea de caracteres, la próxima que se almacena, es decir la novena fila de puntos contando desde la parte superior de la pantalla; luego se almacena la primera fila de la tercera línea de caracteres, y así sucesivamente hasta la primera fila de la línea octava de caracteres. Después se repite la secuencia con la segunda fila de las ocho lineas del carácter y sucesivamente. Y por si esto no fuera suficiente, se repite tres veces la secuencia completa para almacenar los tres bloques de ocho caracteres que componen la pantalla (es decir las 24 líneas, incluyendo el área de entrada).

Esta secuencia de almacenamiento es difícil de entender con una simple descripción, pero si se ejecuta el siguiente programa se verá en «acción».

```
10 FOR i=16384 TO 22527
20 POKE i,255
30 NEXT i
```



Este programa almacena una línea de puntos de tinta (255=BIN 11111111) en cada posición de memoria de la zona de la pantalla, comenzando por el principio (16384) y continuando hacia arriba consecutivamente. El orden de aparición de las líneas horizontales será el que muestra la figura anterior.

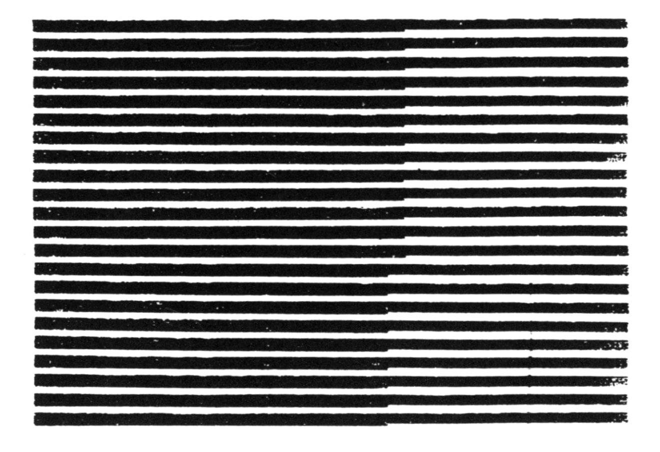
Para sacar algún provecho de saber cómo está almacenada en memoria la pantalla, es preciso poder encontrar cualquier posición de memoria de la pantalla. Es decir, dada una línea, una columna y un número de la fila del carácter, se necesitará una fórmula que facilite la posición de memoria en la que está almacenado. El siguiente programa es más complicado pero se ajusta a la idea.

```
posición de memoria=16384+2048*INT(L)
+32*(L-8*INT(L/8)+256*R+C
```

donde L es el número de la línea, C es el número de la columna, y R es el número de la fila del carácter (todos comenzando desde 0). Por lo tanto, si se quería saber qué configuración de puntos está almacenada en la tercera fila del segundo carácter de la tercera línea, R sería 2, C sería 1 y L

sería 2. Para ver esta fórmula en acción ejecutar el siguiente programa:

```
10 DEF FNs(L,c,r)=16384+2048*INT(L/8)
   +32*(L-8*INT(L/8))+256*r+c
20 CLS
30 FOR n=0 TO 7
40 FOR i=0 TO 31
50 FOR j=0 TO 23
60 POKE FNs(j,i,n),255
70 NEXT j
80 NEXT i
90 NEXT n
```



Este programa llenará la pantalla con líneas horizontales cortas moviéndose verticalmente desde la izquierda. Incluso después de todo este trabajo hay que admitir que la complejidad del formato de almacenamiento de la pantalla en la memoria dificulta el acceso directo a la memoria utilizando las órdenes PEEK y POKE.

Y ahora entramos en el área de memoria de los atributos, que es bastante más fácil de entender. Comenzando en la

22528 y extendiéndose hasta la 23295, hay una posición de memoria para cada posición de carácter de la pantalla. También resulta más fácil la correspondencia entre las posiciones de carácter y las posiciones de memoria. La primera posición de carácter de la línea superior corresponde a la primera posición de memoria, la segunda a la segunda, y así sucesivamente hasta el final de la línea; el primer carácter de la segunda línea corresponde a la posición de memoria 33, y así sucesivamente hasta el final de la pantalla. Para calcular la posición de memoria correspondiente al carácter de la columna C de la línea L, basta ejecutar:

```
22528+32*L+C
```

(Una vez más, no olvidar contar las líneas y las columnas comenzando desde cero). Una posición de memoria almacena los valores actuales de los atributos de una posición de carácter. La forma de almacenamiento de los atributos será analizada más adelante, pero de momento ejecutemos el siguiente programa:

```
10 FOR i=1 TO 32*21
20 PRINT CHR$(INT(RND*26+65));
30 NEXT i
40 FOR x=22528 TO 23295
50 POKE x,INT(RND*256)
60 NEXT x
70 GOTO 40
```

La primera parte del programa, líneas 10 a 30, llena la pantalla de caracteres aleatorios. La segunda parte trasforma el contenido de la memoria de atributos en valores aleatorios, comenzando en la esquina superior izquierda y continuando por toda la pantalla. Hay dos cosas a observar sobre el resultado de este programa. En primer lugar, el orden en que cambian las posiciones de carácter depende de la correspondencia entre las posiciones de memoria y las posiciones de carácter. En segundo lugar, obsérvese que sólo cambian los atributos de lo

que está almacenado en la pantalla; dicho de otra forma, los caracteres presentados en la pantalla no sufren ninguna alteración, sólo cambia su color o su carácter intermitente. A estas alturas podríamos prever que esta característica puede ser de gran utilidad para presentar una imagen en pantalla y cambiar después su color sin tener que volver a imprimir todo de nuevo.

Lo único que nos falta por conocer es la forma exacta en que el contenido del área de atributos de la memoria determina los atributos de las imágenes de la pantalla. Esto se soluciona mediante la siguiente fórmula, que calcula el valor que hay que almacenar en una posición de memoria para generar los atributos deseados.

```
128*FLASH+64*BRIGHT+8*PAPER+INK
```

donde FLASH y BRIGHT son 0 y 1 dependiendo de si la posición va a tener brillo o va a estar en estado intermitente, y PAPER e INK son los códigos de color de los puntos de papel y tinta. Por ejemplo, si se quiere tener un carácter en estado intermitente, con un contraste normal, el papel de color negro y la tinta de color rojo, hay que introducir en la posición de memoria de atributos correspondiente:

```
128*1+64*0+8*0+2
```

es decir 130.

Para hacer la vida más fácil, el Spectrum tiene una función que da el valor del atributo correspondiente a cualquier posición de carácter sin tener que calcular la dirección y ejecutar una orden PEEK. La función es:

```
ATTR(línea,columna)
```

El único problema consiste en que todavía hace falta descifrar el valor que ATTR nos da, por ejemplo, ¿cuál es el color del

papel? Para solucionar este problema podemos utilizar la siguiente lista de instrucciones BASIC:

```
intermitencia=INT(ATTR(línea,col)/128))
brillo=INT((ATTR(línea,col)-INT(ATTR(línea,col)/
128)*128)/64)
papel=INT((ATTR(línea,col)-INT(ATTR(línea,col)/
64)*64)/8)
tinta=ATTR(línea,col)-INT(ATTR(línea,col)/8)*8
```

***SCREEN$ y POINT***

Después de haber analizado el almacenamiento de todo lo relacionado con la imagen de la pantalla, se habrá comprendido que es bastante fácil descubrir de qué color es un carácter. Pero descubrir qué es lo que sale en realidad en la pantalla es bastante más difícil. Incluso si se emplea la fórmula de la dirección de la pantalla, lo único que se puede descubrir escudriñando la pantalla (PEEK) son las características de las ocho filas de puntos que constituyen el carácter. Porque una cosa es conocer la configuración de puntos que forma una letra «A», y otra bien distinta averiguar a qué letra corresponde una configuración de puntos determinada.

Afortunadamente, el BASIC ZX viene en nuestra ayuda con la función SCREEN$. Esta función toma las ocho filas de puntos que aparecen en una posición de carácter e intenta emparejarlos con las configuraciones de puntos que aparecen en la tabla del generador de caracteres (ver el capítulo 7). Si la configuración se empareja con un carácter de la tabla la función SCREEN$ da ese carácter como una cadena, pero en caso contrario la función SCREEN da la cadena nula como respuesta. Adviértase que la única cosa qué importa es la configuración de puntos, y no la forma de crearla; por ejemplo, si se hace la configuración de puntos para la letra «A» utilizando las órdenes PLOT, la función SCREEN$ dará «A» siempre y cuando los puntos se encuentren en el lugar correcto de una posición de carácter. Otra peculiaridad de la forma de

trabajar de la SCREEN$ es que no diferencia entre un carácter y su inverso. Esto significa que un espacio en blanco (CHR$(32), el bloque gráfico sólido [^8] y el bloque gráfico vacío [8] hacen que SCREEN$ dé un espacio en blanco. (En el próximo programa «laberinto» se verá un ejemplo de la utilización de SCREEN$).

También se proporciona una solución al problema más sencillo de definir si un punto determinado de la pantalla es un punto de tinta o de papel.

```
POINT(x,y)
```

Da cero si el punto en cuestión (x,y) es papel y uno si es tinta. En la práctica no es usual que un programa dependa de un punto individual, por lo que POINT no se utiliza tanto como se podría esperar.

**El juego del laberinto**

El juego del laberinto es una versión Spectrum de un juego que está presente en muchos de los microprocesadores populares. La idea básica consiste en tener control sobre los movimientos de un asterisco, que sale de la esquina inferior derecha de la pantalla y debe alcanzar su objetivo, la esquina superior izquierda de la pantalla. Parece fácil, pero... La trampa está en que el camino está bloqueado por una configuración de recuadros oscuros que cambia al azar, y hay que ser hábil en dirigir el asterisco lo más rápidamente posible a través de cualquier abertura antes de que se cierre.

Antes de mirar el listado del programa intentemos pensar cómo nos las ingeniaríamos para escribirlo. Está claro que el problema radica en saber cuándo se va a bloquear el camino. Pero ¿es necesario registrar las coordenadas x e y de cada recuadro oscuro de la pantalla?

```
10 INK 2:PAPER 6:BORDER 1
20 INPUT "Nivel de dificultad 1-9?";d
30 IF d<1 OR d>9 THEN GOTO 20
40 LET d1=d/10
50 CLS
60 GOSUB 100
70 GOSUB 300
80 GOSUB 500

100 POKE USR "a"+0,BIN 01010101
110 POKE USR "a"+1,BIN 10101010
120 POKE USR "a"+2,BIN 01010101
130 POKE USR "a"+3,BIN 10101010
140 POKE USR "a"+4,BIN 01010101
150 POKE USR "a"+5,BIN 10101010
160 POKE USR "a"+6,BIN 01010101
170 POKE USR "a"+7,BIN 10101010
180 RETURN

200 LET c$=SCREEN$(y,x)
210 RETURN

300 REM setup maze
310 GOSUB 1000
320 FOR i=1 TO 20*d+40
330 LET x=RND*29+1
340 LET y=RND*19+1
350 PRINT AT y,x;"[a]";
360 NEXT i
370 PRINT AT 1,1;"$";
380 PRINT AT 20,30;"*";
400 RETURN

500 LET xc=30
510 LET yc=20
520 LET m=0
530 LET x=xc
540 LET y=yc
```

```
550 LET a$=INKEY$
560 GOSUB 900
570 IF a$="" THEN GOTO 550
580 IF a$="5" THEN LET x=xc-1
590 IF a$="8" THEN LET x=xc+1
600 IF a$="6" THEN LET y=yc+1
610 IF a$="7" THEN LET y=yc-1
620 GOSUB 200
630 IF c$="$" THEN GOTO 800
640 IF c$<>" " THEN BEEP .1,10:GOTO 530
650 PRINT AT yc,xc;" ";
660 LET xc=x
670 LET yc=y
680 PRINT AT yc,xc;"*";
690 LET m=m+1
700 GOTO 530

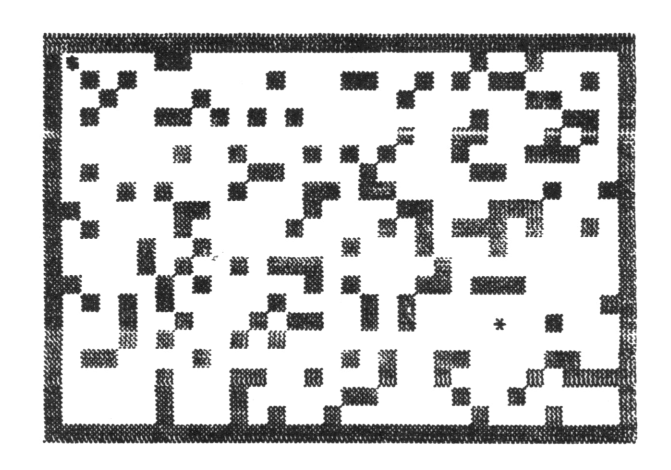
800 CLS
810 PRINT AT 0,2;"Has tardado ";m;" movimientos"
820 PRINT "Otro juego s/n"
830 INPUT a$
840 IF a$(1)="s" THEN RUN
850 IF a$(1)<>"n" THEN GOTO 820
860 STOP

900 IF RND>d1 THEN RETURN
910 LET c$=" "
920 IF RND>.5 THEN LET c$="[a]"
930 PRINT AT RND*19+1,RND*29+1;c$
940 PRINT AT 1,1;"$";
950 RETURN

1000 FOR i=0 TO 31
1010 PRINT AT 0,i;INK 0;"[a]";AT 21,i;"[a]";
1020 NEXT i
1030 FOR i=0 TO 21
1040 PRINT AT i,0;INK 0;"[a]";AT i,31;"[a]";
1050 NEXT i
1060 RETURN
```

El programa comienza solicitando el nivel de dificultad, d, que determina la cantidad de recuadros que bloquearán el camino. La subrutina 300 utiliza esta información para construir el laberinto inicial. Los detalles son sencillos: En primer lugar, una llamada a la subrutina 1000 dibuja un marco alrededor del laberinto utilizando los caracteres gráficos definidos por la subrutina 100; después, una orden PRINT AT imprime el bloque gráfico [A] en puntos aleatorios de la pantalla; y antes de abandonar la subrutina, se imprime un signo «$» en la esquina superior izquierda de la pantalla para representar la meta que debe alcanzar el asterisco, impreso en la esquina inferior derecha de la pantalla.

El juego propiamente dicho comienza con una llamada a la subrutina 500. Tras la inicialización, la línea 550 supervisa las teclas de flecha; la pulsación de las teclas de flecha actualiza los valores de xc y yc (la posición actual del asterisco) convirtiéndolos en x e y (la posición donde se lleva el asterisco), que dependerán de la tecla que se haya pulsado. El siguiente problema consiste en decidir si la posición a la que se envía el asterisco es legal o no; si contiene sólo un carácter en blanco es legal llevar ahí el asterisco, pero si contiene cualquier otro carácter habrá que rechazar el movimiento. La línea 620 llama a la subrutina 200 para que examine la pantalla y averigue qué hay en la posición x,y (mediante la función SCREEN$); la línea 200 pone en la variable c$ el carácter que hay en esa posición y si resulta ser una «$» el control salta a la línea 800 para poner fin al juego. Si no se da esta situación, la línea 640 comprueba que c$ es un espacio en blanco, y si no lo es rechaza el movimiento con un BEEP. Pero si es un espacio en blanco, la línea 650 borra la posición anterior del asterisco y la línea 680 imprime el nuevo asterisco. La línea 700 incrementa el contador de movimientos m y repite la lógica del movimiento. Los únicos detalles que no hemos analizado hasta ahora son: la subrutina 900, que pone y quita los obstáculos, y las líneas 800 a 860, que imprimen el resultado de un juego y preguntan si se quiere jugar de nuevo; ambas deberían resultar fáciles de comprender. Adviértase que la



utilización de la SCREEN$ para descubrir qué hay en la posición a imprimir propuesta impide también que el asterisco sea «dirigido» fuera de la pantalla, sin necesidad de utilizar instrucciones IF adicionales. Debido a que el laberinto está rodeado por un marco de caracteres, y no por espacios en blanco, la lógica del movimiento no permitirá cruzarlo.

Después de haber estudiado las técnicas presentadas en este capítulo, se estará en condiciones de utilizar PEEK, POKE y SCREEN$ para generar con buenos resultados gráficos animados. Un buen ejercicio sería cambiar el programa de squash mostrado anteriormente utilizando la función SCREEN$ para detectar si la bola va a golpear el bate o las paredes laterales.

**Corrimiento de gráficos**

Hasta ahora hemos acumulado una amplia gama de métodos para generar gráficos en movimiento pero todavía hay un problema de difícil solución: mover varias cosas al mismo tiempo. Es imposible mover más de una o dos cosas a cual-

quier velocidad en el Spectrum utilizando el BASIC. Sin embargo hay una excepción. Ejecutar el siguiente programa.

```
10 CLS
20 POKE 23692,255
30 PRINT AT 21,RND*31;"*"''
40 GOTO 20
```

La línea 30 provoca el corrimiento de la pantalla mediante el doble apóstrofe colocado al final, y mueve toda la pantalla verticalmente una línea. El tiempo que tarda no depende del número de cosas que se mueven. La línea 20 es necesaria para suprimir el mensaje de corrimiento (Scroll?) que aparece (ver el Capítulo 7 para más detalles). La dificultad del corrimiento de gráficos no es mover cosas sino hacer que se estén quietas. Si se imprime algo en la línea superior después de un corrimiento parecerá estar parado en medio de una corriente de asteriscos en movimiento. Añadir:

```
35 PRINT AT 0,16;"V";
```

al programa anterior y se tendrá el comienzo del juego «pilotar una nave espacial por la galaxia».

**El juego de la carrera de esquí**

Para mostrar el poder del corrimiento de gráficos, ejecútese el siguiente programa de la carrera de esquí:

```
10 RAND 0
20 LET b$="<>"
30 CLS
40 DIM c(26,2)
50 GOSUB 5000
60 GOSUB 1000
70 LET t=0
80 LET p=0
```

*(Tener presente que es importante escribir en la línea 20 etc, un símbolo «menos que» seguido a continuación de un símbolo «mayor que» en lugar de un «signo de desigualdad»).*

```
90 LET x=16
100 GOSUB 2000
110 GOTO 4000

1000 FOR i=1 TO 25
1010 LET c(i,1)=INT(RND*5)+5
1020 LET c(i,2)=INT(RND*25)+4
1030 NEXT i
1040 LET c(1,1)=21
1050 RETURN

2000 POKE 23692,255:PRINT AT 21,c(1,2);">";
2010 LET k=1
2020 LET j=0
2030 LET m=0
2040 LET i=2
2050 LET s=0
2060 LET s=s+1
2070 LET t=t+1
2080 PRINT AT 21,0 ''
2090 GOSUB 3000
2100 IF s<>c(i,1) THEN GOTO 2060
2110 POKE 23692,255: PRINT AT 21,c(i,2);b$(j+1);
2120 LET j=NOT j
2130 LET i=i+1
2140 IF i<26 THEN GOTO 2050
2150 FOR i=1 TO 23
2160 LET t=t+1
2170 POKE 23692,255:PRINT AT 21,0 ''
2180 GOSUB 3000
2190 NEXT i
2200 RETURN

3000 LET a$=INKEY$
3010 IF a$="5" THEN LET x=x-1
3020 IF a$="8" THEN LET x=x+1
3030 PRINT AT 0,x;"*";
3040 IF t<>c(m+1,1) THEN RETURN
```

```
3050 LET t=0
3060 LET m=m+1
3070 LET k=NOT k
3080 IF NOT k AND (x-c(m,2)) >0 THEN RETURN
3090 IF k AND (x-c(m,2)) <0 THEN RETURN
3100 LET p=p+1
3110 BEEP .05,20:PRINT "h"
3120 RETURN

4000 PRINT
4010 PRINT "Has golpeado ";p;" puertas"
4020 PRINT "¿Quieres intentarlo de nuevo?";
4030 INPUT a$
4040 PRINT a$
4050 IF a$(1)="n" THEN STOP
4060 IF a$<>"s" THEN GOTO 4020
4070 PRINT "La misma pista?";
4080 INPUT a$
4090 PRINT a$
4100 IF a$(1)="s" THEN GOTO 70
4110 IF a$(1)<>"n" THEN GOTO 4070
4120 GOTO 60

5000 CLS
5010 PRINT
5020 PRINT TAB 8;"S K I  R U N"
5030 PRINT TAB 8;"-------------"
5040 PRINT
5050 PRINT "Tienes que descender por una pista"
5060 PRINT "de 25 banderas"
5070 PRINT
5080 PRINT "Debes pasar por la izquierda de"
5090 PRINT "< y por la derecha de >"
5100 PRINT
5110 PRINT "Te puedes mover a la derecha y"
5120 PRINT "a la izquierda pulsando las teclas de flechas"
5130 PRINT "(5 y 8)"
5140 PRINT
```

```
5150 PRINT "Pulsar cualquier tecla para comenzar"
5160 IF INKEY$=" " THEN GOTO 5160
5170 CLS
5180 RETURN
```

Aunque la idea en que se basa este juego es bastante simple y se funda en el corrimiento de gráficos, son necesarias algunas formas ingeniosas de «contabilidad» para saber en qué puntos de la pantalla están situadas las cosas. En primer lugar, la subrutina 5000 da las reglas del juego y después la subrutina 1000 construye al azar una pista, generando dos números aleatorios: la distancia (el número de corrimientos) comprendida entre cada puerta, y su posición horizontal. El juego comienza con una llamada a la subrutina 4000, que imprime la primera puerta y comienza a correr la pantalla en dirección al esquiador, respresentado por un asterisco. La parte difícil consiste en que después de un cierto número de corrimientos, que contiene c(2,1), se imprime la siguiente puerta, y después de 21 corrimientos (no olvidar, que hay 22 líneas útiles en la pantalla) la primera puerta llega al esquiador. En este momento se supervisa la posición del esquiador para ver si está en el lado correcto de la puerta. A estas alturas del libro se debe saber lo suficiente para ver que siguiendo esta lógica es posible calcular el momento en que las puertas llegan al esquiador y utilizar este dato a fin de llevar la cuenta del número de puertas que el esquiador ha pasado correctamente. Los detalles relativos a este punto (líneas 2000 a 3120) son más fáciles de programar que de explicar, por lo que se dejan en manos del lector.

**Resumen**

Las técnicas descritas en este capítulo pueden servir para generar algunos gráficos muy ingeniosos. Por ejemplo, se podrían combinar los métodos de corrimiento de gráficos con gráficos SCREEN$, o utilizar las órdenes PEEK y POKE en el área de atributos para generar colores en movimiento.

usues

- El increíble Spectrum ZX ofrece a sus usuarios un campo de acción virtualmente ilimitado. Permite un uso versátil del color, ofrece gráficos de alta y baja resolución, y además añade sonido. El resultado es la posibilidad de escribir programas BASIC prácticos y excitantes.

- Pero existe un problema: para que el Spectrum haga cosas ingeniosas hay que poner en práctica algo más: una cosa es haber aprendido a usar las órdenes del Spectrum, y otra muy diferente ser capaz de combinarlas en programas que hagan exactamente lo que se desea. Y ese es precisamente el objetivo de este libro: enseñar todos los secretos necesarios para hacer programas con el Spectrum.

- Este libro comprende los siguientes capítulos: 1. Un primer contacto con el Spectrum; 2. Gráficos de baja resolución; 3. La diversión del azar; 4. Gráficos de alta resolución; 5. Sonido; 6. Gráficos en movimiento; 7. PEEK y POKE; 8. Una sensación de tiempo; 9. Cadenas y Palabras; 10. Gráficos avanzados.

- Un libro, pues, esencial para todos los usuarios de ordenadores Spectrum, tanto para los principiantes como para los ya expertos.

